

THESIS

HYPOTHESIS-BASED MACHINE LEARNING FOR DEEP-WATER CHANNEL SYSTEMS

Submitted by

Noah Francis Ryoichi Vento

Department of Geosciences

In partial fulfillment of requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2020

Master's Committee:

Advisor: Lisa Stright

Michael Ronayne
Charles Anderson

Copyright by Noah Francis Ryoichi Vento 2020

All Rights Reserved

ABSTRACT

HYPOTHESIS-BASED MACHINE LEARNING FOR DEEP-WATER CHANNEL SYSTEMS

Machine learning algorithms are readily being incorporated into petroleum industry workflows for use in well-log correlation, prediction of rock properties, and seismic data interpretation. However, there is a clear disconnect between sedimentology and data analytics in these workflows because sedimentologic data is largely qualitative and descriptive. Sedimentology defines stratigraphic architecture and heterogeneity, which can greatly impact reservoir quality and connectivity and thus hydrocarbon recovery. Deep-water channel systems are an example where predicting reservoir architecture is critical to mitigating risk in hydrocarbon exploration. Deep-water reservoirs are characterized by spatial and temporal variations in channel body stacking patterns, which are difficult to predict with the paucity of borehole data and low quality seismic available in these remote locations. These stacking patterns have been shown to be a key variable that controls reservoir connectivity.

In this study, the gap between sedimentology and data analytics is bridged using machine learning algorithms to predict stratigraphic architecture and heterogeneity in a deep-water slope channel system. The algorithms classify variables that capture channel stacking patterns (i.e., channel positions: axis, off-axis, and margin) from a database of outcrop statistics sourced from 68 stratigraphic measured sections from outcrops of the Upper Cretaceous Tres Pasos Formation at Laguna Figueroa in the Magallanes Basin, Chile. An initial hypothesis that channel position could be predicted from 1D descriptive sedimentologic data was tested with a series of machine learning algorithms and classification schemes. The results confirmed this hypothesis as complex

algorithms (i.e., random forest, XGBoost, and neural networks) achieved accuracies above 80% while less complex algorithms (i.e., decision trees) achieved lower accuracies between 60%-70%. However, certain classes were difficult for the machine learning algorithms to classify, such as the transitional off-axis class. Additionally, an interpretive classification scheme performed better (by around 10%-20% in some cases) than a geometric scheme that was devised to remove interpretation bias. However, outcrop observations reveal that the interpretive classification scheme may be an over-simplified approach and that more heterogeneity likely exists in each class as revealed by the geometric scheme. A refined hypothesis was developed that a hierarchical machine learning approach could lend deeper insight into the heterogeneity within sedimentologic classes that are difficult for an interpreter to discern by observation alone. This hierarchical analysis revealed distinct sub-classes in the margin channel position that highlight variations in margin depositional style. The conceptual impact of these varying margin styles on fluid flow and connectivity is shown.

ACKNOWLEDGEMENTS

This work is a culmination of geologic research conducted by the Chile Slope Systems (CSS) Joint Industry Project, which is a collaboration between the University of Calgary, Colorado State University, Virginia Tech, and industry partners: Chevron, Repsol, Hess, Nexen/CNOOC, ConocoPhillips, BHP Billiton, Anadarko, Equinor, Petrobras, and Shell. Fieldwork, data collection, and interpretation were performed by Brian Romans, Steve Hubbard, Ryan Macauley, Sean Fletcher, and Sarah Southern. The starter neural network code in Python was written by Chuck Anderson, and the MATLAB digitizer code was written by Zane Jobe. Special thanks to both of them for their discussions, expertise, and guidance on bridging the gap between data science and geology. Lastly, thanks to Lisa Stright for her continuous support and assistance throughout this project. Without her insight, this project would not have been possible.

TABLE OF CONTENTS

ABSTRACT.....	ii
ACKNOWLEDGEMENTS	iv
CHAPTER 1: RESEARCH MOTIVATION.....	1
1.1 Introduction.....	1
1.2 Study Overview	4
1.3 Machine Learning in Petroleum Geoscience	6
1.4 Thesis Format.....	8
CHAPTER 2: GEOLOGIC BACKGROUND AND DATABASE INTRODUCTION	10
2.1 Geologic Setting.....	10
2.2 Laguna Figueroa Database.....	15
2.2.1 Channel Outcrop Statistics.....	15
2.2.1.1 Facies Association Proportions.....	16
2.2.1.2 Net.....	17
2.2.1.3 Gross	17
2.2.1.4 Net-To-Gross	17
2.2.1.5 Drape Thickness.....	17
2.2.1.6 Bed Statistics (Count, Minimum, Median, and Maximum).....	17
2.2.1.7 Amalgamation Ratio	18
2.2.1.8 Grain Size Distributions and P10, P50, and P90 Statistics	18
2.2.2 Classification Schemes	18
2.2.2.1 Facies-Driven	19
2.2.2.2 Geometric.....	19
2.2.3 Generation of Hypotheses.....	20
CHAPTER 3: MACHINE LEARNING OVERVIEW.....	26
3.1 Machine Learning Algorithms	26
3.1.1 Unsupervised Learning	26
3.1.1.1 Feature Importance with Principal Component Analysis	26
3.1.1.2 Clustering Analysis with K-Means.....	27
3.1.2 Supervised Learning	27
3.1.2.1 Decision Trees	27
3.1.2.2 Discriminant Analysis.....	28

3.1.2.3 Naïve Bayes	28
3.1.2.4 Support Vector Machines	29
3.1.2.5 K-Nearest Neighbors	29
3.1.2.6 Ensemble Classifiers	30
3.1.2.7 K-Fold Cross-Validation.....	32
3.1.3 Deep Learning.....	34
3.1.3.1 Neural Networks	34
3.2 Evaluation Metrics	37
3.2.1 Validation Accuracy	37
3.2.2 Confusion Matrix	37
3.2.3 Precision.....	37
3.2.4 Recall	38
3.2.5 F1 Score	38
CHAPTER 4: EVALUATING MACHINE LEARNING ALGORITHMS FOR PREDICTION OF CHANNEL POSITION	39
4.1 Methodology	39
4.2 Results.....	39
4.2.1 Unsupervised Learning Results	39
4.2.1.1 Feature Importance with Principal Component Analysis	39
4.2.1.2 Clustering Analysis with K-Means.....	43
4.2.2 Supervised Learning Results.....	44
4.2.2.1 Unsupervised vs. Supervised Learning	45
4.2.2.2 Facies-Driven vs. Geometric Classification Schemes	48
4.2.2.3 Two Positions vs. Three Positions	52
4.2.2.4 Individual Channel Positions	52
4.3 Refinement of Hypotheses	54
CHAPTER 5: HIERARCHICAL MACHINE LEARNING ANALYSIS.....	57
5.1 Methodology	57
5.2 Results.....	57
CHAPTER 6: DISCUSSION.....	64
6.1 Efficacy of Machine Learning Algorithms in Channel Outcrop Analysis	64
6.2 Variations in Intra-Channel Fill and Impacts on Fluid Flow and Connectivity	67
CHAPTER 7: CONCLUSIONS AND FUTURE WORK.....	69
7.1 Conclusions.....	69
7.2 Future Work	70

7.2.1 Modeling Channel Stacking Scenarios	70
7.2.2 Classifying Channel Position from Well-Log Data	70
7.2.3 Automatic Detection of Channel Boundaries	70
7.2.4 Data Augmentation	71
7.2.5 Testing on a Different Deep-Water Channel System	71
REFERENCES	74
APPENDIX A: LAGUNA FIGUEROA DATABASE	82
APPENDIX B: PYTHON CODES.....	106
APPENDIX C: MACHINE LEARNING RESULTS.....	138

CHAPTER 1: RESEARCH MOTIVATION

1.1 Introduction

Deep-water channel systems transport sediment from the continental shelf to the abyssal plain. Over time, this process of sediment transport results in channel geobodies that stack both laterally and vertically, creating ideal reservoirs for hydrocarbon storage. A channel geobody—also referred to as a channel element—is the fundamental architectural unit in deep-water channel systems and consists of an incisional channel-form surface and its sediment fill (Figure 1B; McHargue et al., 2011). Channel geobodies that are genetically related and stack in a consistent pattern form a channel complex (Figure 1A; McHargue et al., 2011). Multiple genetically related channel complexes create a channel complex set (Figure 1A; McHargue et al., 2011).

Channel geobodies are commonly divided into different segments based on the position within the body—axis, off-axis, and margin—and sediment fill can vary across these different positions due to differences in flow energy (Figure 1B; Southern et al., 2017). The channel axis, which represents the deepest portion of a channel geobody, experiences higher-energy flows resulting in deposition of coarser sediment and higher rates of sandstone bed amalgamation (McHargue et al., 2011; Southern et al., 2017). As flow energy wanes laterally, sediment grain size decreases and facies become less amalgamated and more heterolithic, resulting in off-axis and margin channel positions with limited sandstone bed amalgamation (McHargue et al., 2011; Southern et al., 2017).

The petroleum industry is interested and invested in better understanding these architectural variations and their subsequent effects on channel stacking patterns in deep-water

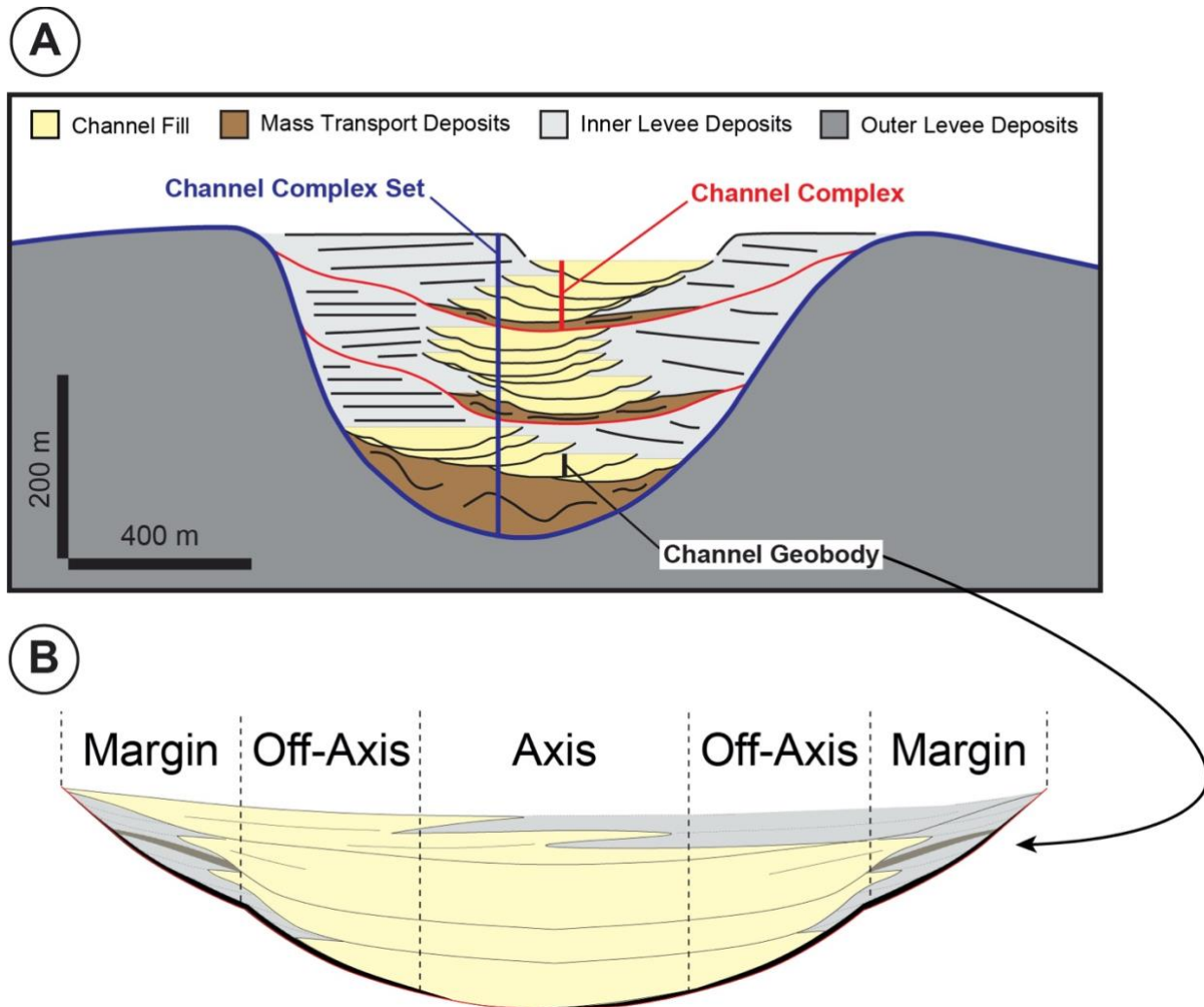


Figure 1. (A) Slope channel hierarchy (modified from Daniels (2019)). (B) Conceptual model of channel geobody with channel positions—axis, off-axis, and margin—labeled.

channel systems to improve reservoir characterization (Deptuck et al., 2007) and thus reduce risk during exploration, development, and production. Characterizing intra- and inter-channel architecture is critical because it can have significant impacts on reservoir connectivity (Barton et al., 2010; Alpak et al., 2013; Meirovitz et al., *accepted pending revision*). However, this stratigraphic architecture can be difficult to interpret in exploration-scale seismic data (Chopra et al., 2006; Hart, 2013; Pemberton et al., 2018).

The use of outcrop studies as analogs can help to improve our understanding of facies distribution and heterogeneity with bed- to geobody-scale field observations and measurements not commonly available in subsurface data (Macauley and Hubbard, 2013). A wealth of quantitative data and statistics can be extracted from outcrop analogs and used to test and guide subsurface interpretation workflows (Southern et al., 2017; Daniels et al., 2019). These quantitative data can be explored with automated analytical techniques, such as machine learning and deep learning, which have had a renaissance recently due to improvements in technology and their usefulness for processing large datasets efficiently (Mohammadpoor and Torabi, 2019).

Machine learning is a subset of artificial intelligence that allows computing systems to learn and improve without being explicitly programmed (Samuel, 1959). It typically consists of two steps: training and classification. First, a portion of the data is used to train the machine learning algorithms to generate classifications. Then, the algorithms are tested on the remaining data. The algorithms minimize misclassifications by adapting to the properties of the data over numerous iterations or epochs. Common subdivisions of machine learning include supervised, unsupervised, and deep learning. In supervised learning, training samples are classified or provided with their known labels (Kotsiantis et al., 2006). Conversely, in unsupervised learning, algorithms are provided with unlabeled data and must generate classifications solely based on patterns and trends (Jain et al., 1999), often referred to as clustering. The final subdivision, deep learning, makes use of supervised and unsupervised learning techniques but primarily focuses on the implementation of neural networks, which are algorithms that are modeled after the human brain (McCulloch and Pitts, 1943). Neural networks consist of multiple interconnected processing layers composed of individual units called neurons, which activate based on weights and biases to generate classifications. Neural networks have been revolutionary in processing image, video,

speech, text, and audio data (Lecun et al., 2015) and are proving to be useful in interpreting geologic data as well (Cheng et al., 2019; Mohammadpoor and Torabi, 2019; Na and Fox, 2019).

1.2 Study Overview

In this study, we utilize an outcrop database of a deep-water slope channel system to: 1) test the efficacy of machine learning algorithms in adequately predicting deep-water stacking patterns from 1D borehole data; and 2) implement a hierarchical machine learning workflow to explore the differences in intra-channel architecture and how it is interpreted, and highlight its potential impacts on reservoir connectivity and fluid flow. The database contains over 3,400 meters in 68 measured sections from deep-water slope channel strata in the Tres Pasos Formation at Laguna Figueroa in the Magallanes Basin, Chile (Appendix A; Fletcher, 2013; Macauley and Hubbard, 2013; Southern et al., 2017). The measured section data includes grain size, bed thickness, and sedimentary structure information, which were used to interpret facies associations and stratal packages within the upper and lower channel complex sets at Laguna Figueroa. From these measurements and interpretations, statistics including net-to-gross, channel thickness, facies proportions, grain size distributions, and amalgamation ratio were calculated for individual channel geobodies and labeled using different classification schemes for channel position. The first classification scheme was an interpretive, facies-driven scheme that separates channels into positions based on expert geologic interpretations made in the field. The second scheme was an objective, geometric scheme that separates channels into positions based on height above the base of a channel geobody. This scheme was implemented to account for any internal bias generated by the interpretations made in the facies-driven scheme.

After the database was established, the gamut of machine learning algorithms—unsupervised principal component analysis (PCA) and K-means, 26 supervised learners, and a

deep learning neural network—was used to analyze the statistics and generate classifications of channel position for the different classification schemes while testing hypotheses. The first hypothesis was that machine learning algorithms can be useful for predicting sedimentologic classes (i.e., channel position axis, off-axis and margin). Observations show that statistics for axis and margin are distinct groups and end members, while off-axis statistics appear to be more intermediate and transitional. The ability of machine learning to differentiate distinct heterogeneous groups in the margin position, facies that are critical to controlling fluid flow between channel geobodies (Jackson et al., 2019; Meirovitz et al., *accepted pending revision*) is hindered by potential overlapping characteristics with axis and off-axis facies. Furthermore, it is suspected that facies-driven classification schemes are biased by sedimentologic similarities that do not necessarily define position in a channel geobody.

Therefore, a refined hypothesis was devised that a hierarchical machine learning approach could better categorize sub-classes of the data and lend deeper insight into sedimentologic classes difficult for an interpreter to discern by observation alone and reveal higher degrees of heterogeneity that are observed in outcrop. The field observations are that different styles of intra-channel fill architecture exist due to variations in geological genesis, characteristics, and trends in the outcrop data (Southern et al., 2017). This hierarchical analysis was designed to test for such sub-classes observed in the field, but difficult to interpret from the data itself.

The results of this study are analyzed to discuss: 1) the use of machine learning algorithms for outcrop- and core-based studies; 2) differences in intra-channel fill architecture and heterogeneity; 3) the implications for the impact of channel position prediction on fluid flow and reservoir connectivity. Additionally, the results motivate future work incorporating machine

learning workflows with diverse datasets and projects to improve predictions in geo-modeling and well-log analysis.

1.3 Machine Learning in Petroleum Geoscience

The term “Big Data” refers to datasets with such high volume, variety, velocity, and veracity that specialized analytics are required to process them efficiently (Mohammadpoor and Torabi, 2019). Big Data has become commonplace in many different occupations and disciplines over the past decade (Chen et al., 2012; Martin-Sanchez and Verspoor, 2014; Mohammadpoor and Torabi, 2019). With innovations in data acquisition, new technologies, and complex problems, the oil and gas (O&G) industry is relying on automated analytical techniques, such as artificial intelligence and machine learning, to process, analyze, and extract useful information from Big Data in both upstream and downstream projects (Hassani and Silva, 2018; Mohammadpoor and Torabi, 2019). This study focuses on the application of machine learning to upstream projects, more specifically, exploration and development of deep-water O&G reservoirs.

Upstream O&G studies of machine learning have commonly applied these analytical techniques to seismic data for stratigraphic and structural interpretation and facies prediction, and well-log data for correlation and facies prediction. These are common components of reservoir characterization and modeling, which guide resource estimation, well placement planning, and reservoir performance forecasting (Bubnova et al., 2019). Seismic data provide the general subsurface geology and architecture of an area, allowing interpreters to identify geologic structures and seismic facies (McHargue and Webb, 1986; Nader et al., 2016). Recent improvements in data acquisition methods and technologies have resulted in a boost in seismic data availability, which has resulted in the increased usage of machine learning algorithms (Mohammadpoor and Torabi, 2019). These algorithms have been used to identify subsurface faults from synthetic seismic

volumes (Huang et al., 2017) and also predict seismic facies from 3D broadband seismic reflectivity data with accuracies of up to 98.3% (Wrona et al., 2018).

Although the benefits of seismic data to any petroleum industry project are profound, exploration-scale seismic data often does not have the resolution to detect complex heterogeneity and stratigraphy in the subsurface (Hart, 2013; Pemberton et al., 2018). To mitigate this issue, well-logs are typically used to interpret facies and stratal packages (Van Wagoner et al., 1990; Bubnova et al., 2019). Well-logs record the petrophysical responses of subsurface rocks to various forms of measurements (Asquith et al., 2004). Interpretation of these responses is crucial for identifying facies and thus correlating subsurface stratigraphy throughout a potential reservoir or field. However, some fields, particularly older fields that have been developing for a long time, have numerous wells and logs making it difficult and time-consuming to interpret each thoroughly. Over the past few years, machine learning algorithms have been successful at aiding in well-log correlation (Brazell et al., 2019) and facies prediction even in wells with diverse lithologies (Hall, 2016; Bestagini et al., 2017).

Despite these diverse applications of data analytics to the O&G industry, machine learning algorithms have seldom been used on physical geologic data, such as core, which directly represent stratigraphic architecture and thus the complex heterogeneity that impacts reservoir quality and flow rates. The lack of machine learning studies in this realm of upstream O&G is due, in part, to the inherently qualitative and subjective nature of interpreting core data, whereas interpretation of seismic and well-log data often appears to be more quantitative because the data is a measured response from the earth model. However, the addition of physical geologic data and interpretations can benefit machine learning analyses and improve predictions. In some cases, accuracies of over 83% have been achieved when incorporating detailed petrographic analyses of thin sections and

textural information to well-log studies for facies prediction (Saporetti et al., 2018). Nevertheless, in a machine learning workflow, it is imperative to have dependable classifications or labels for the sample data to ensure that uncertainty and error are not being added into the model from the beginning. Core data is one-dimensional, sparse and often biased due to drilling locations, which makes it difficult to predict lateral and vertical trends, thus making it impossible to obtain full certainty in classifications of stratigraphic architecture in a system. However, this study is novel in that regard as it utilizes measured sections, which are analogous to 1D borehole and core data, to predict stratigraphic architecture in a deep-water slope channel system. This is made possible because the deep-water outcrops of the Tres Pasos Formation at Laguna Figueroa in the Magallanes Basin are world-class. Specifically, outcrops of the Tres Pasos Formation at Laguna Figueroa provide high-quality 2D to 3D exposure of channelized turbidite deposits (Macauley and Hubbard, 2013; Southern et al., 2017; Pemberton et al., 2018; Jackson et al., 2019), which allows for a high degree of confidence in their geologic interpretation and classification. These outcrops have been studied extensively by a multitude of researchers over a decade of field seasons (Hubbard et al., 2010; Romans et al., 2011; Macauley and Hubbard, 2013; Hubbard et al., 2018; Pemberton et al., 2018; Daniels et al., 2019), culminating in a firm understanding of the entire depositional system and a refined and robust interpretation of the stratigraphic correlation from element- to system-scale at Laguna Figueroa. This interpretation is the foundation of the extensive database used to train and test the machine learning algorithms in this study.

1.4 Thesis Format

The remainder of this thesis is partitioned into six main chapters (Chapters 2-7). Chapter 2 discusses the geologic background of the Magallanes Basin, how the database for this study was constructed and labeled based on measured section data from Laguna Figueroa, and how the

database was analyzed to generate hypotheses for the preliminary machine learning analyses. Chapter 3 features background information on the machine learning algorithms and methods that were used in the analyses—as well as the evaluation metrics that were used to assess their performance. Chapter 4 presents the methodology and results for the preliminary machine learning analyses and motivates a refinement of hypotheses for the hierarchical machine learning workflow performed in Chapter 5. Chapter 5 details the methodology for the hierarchical machine learning workflow and its results. Chapter 6 discusses the results from Chapters 4 and 5 and their implications for exploration projects. Chapter 7 concludes the work performed in this study and presents future projects that can use machine learning algorithms and data analytics to extract value from 1D borehole data.

CHAPTER 2: GEOLOGIC BACKGROUND AND DATABASE INTRODUCTION

2.1 Geologic Setting

The Magallanes Basin in southern Chile records the tectonic evolution of southernmost South America throughout the Cretaceous (Fildani and Hessler, 2005; Romans et al., 2011; Daniels et al., 2019). Extension in the Jurassic related to the breakup of Gondwana resulted in the formation of the Rocas Verdes backarc basin, and subsequent compression related to the Andean Orogeny resulted in the closure of the Rocas Verdes Basin and the formation of the Magallanes retroarc foreland basin (Dalziel et al., 1974; Wilson, 1991; Fildani and Hessler, 2005; Fosdick et al., 2011; Romans et al., 2011).

The Tres Pasos Formation is a progradational slope system (Romans et al., 2009; Hubbard et al., 2010; Romans et al., 2011; Macauley and Hubbard, 2013) that represents an up to 1500 m thick section of the overall 4-5 km of deep-water fill that makes up the Magallanes Basin (Covault et al., 2009; Romans et al., 2009; Romans et al., 2011; Macauley and Hubbard, 2013). This section is composed of sandstone-rich channels and mudstone-rich mass transport deposits (MTDs) that overlie the muddy and conglomeratic deposits of the Punta Barrosa and Cerro Toro Formations and are genetically linked to the shallow marine deltaics of the Dorotea Formation above (Figure 2; Romans et al., 2009; Romans et al., 2011; Macauley and Hubbard, 2013). Notable outcrops of the Tres Pasos Formation are located in the Última Esperanza District of Chile (Figure 2; Hubbard et al., 2010; Daniels et al., 2019).

Laguna Figueroa is a 2.5 km long, 300 m thick section of the progradational slope system located north of the town of Puerto Natales (Figure 2). The outcrops at Laguna Figueroa show high-quality 2D to 3D oblique-dip oriented exposure of channelized turbidite deposits (Macauley

and Hubbard, 2013; Southern et al., 2017; Pemberton et al., 2018; Jackson et al., 2019). These deposits stack to form two channel complex sets—Lower Figueroa and Upper Figueroa. Each channel complex set is composed of multiple channel complexes and associated channel bodies (McHargue et al., 2011), which were interpreted extensively by Macauley and Hubbard, 2013. The lowermost channel complex set is composed of three channel complexes and twelve distinct channel bodies, and the uppermost is composed of four channel complexes and thirteen distinct channel bodies (Figure 3).

Macauley and Hubbard (2013) interpreted facies associations for the Laguna Figueroa outcrops [FA1-FA4] (Figure 4C), which include: thick-bedded, highly amalgamated sandstone (FA1); thick- to thin-bedded, semi-amalgamated sandstone and siltstone (FA2); thick- to thin-bedded, largely non-amalgamated sandstone and siltstone (sandstone dominated) (FA3); and medium- to very-thin bedded largely non-amalgamated sandstone and siltstone (siltstone dominated) (FA4). These facies associations as well as descriptions from the measured sections with cm-scale resolution provide the foundation for the training database used in this study.

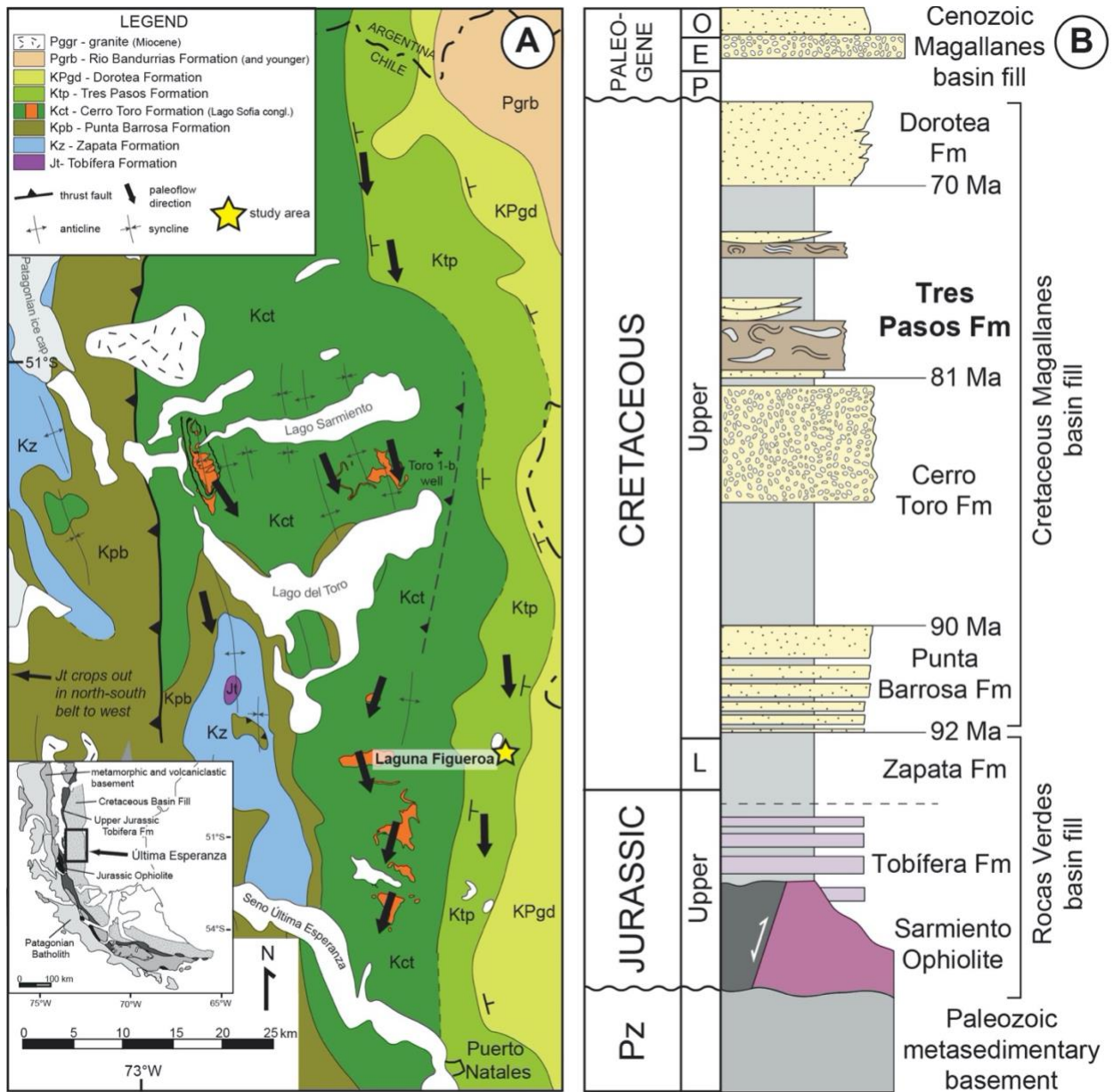


Figure 2. (A) Geologic map of Última Esperanza District in southern Chile (modified from Romans et al., (2011); originally adapted from Wilson (1991) and Fosdick et al., (2011)) depicting the primary formations of the Magallanes Basin. The Late Cretaceous-age strata that composes the basin gets older moving to the east with a paleoflow direction of south to southeast along the axis of the elongate basin. The formation of interest for this study, Tres Pasos Fm. (Ktp), is located to the east of the Cerro Toro Fm. (Kct) and west of the Dorotea Fm. (Kpgd). The study area, Laguna Figueroa, is located north of Puerto Natales as represented by the star marker. (B) Stratigraphic column of the Magallanes Basin (modified from Daniels et al., (2018), GSA Bulletin) with the Tres Pasos Fm. bolded.

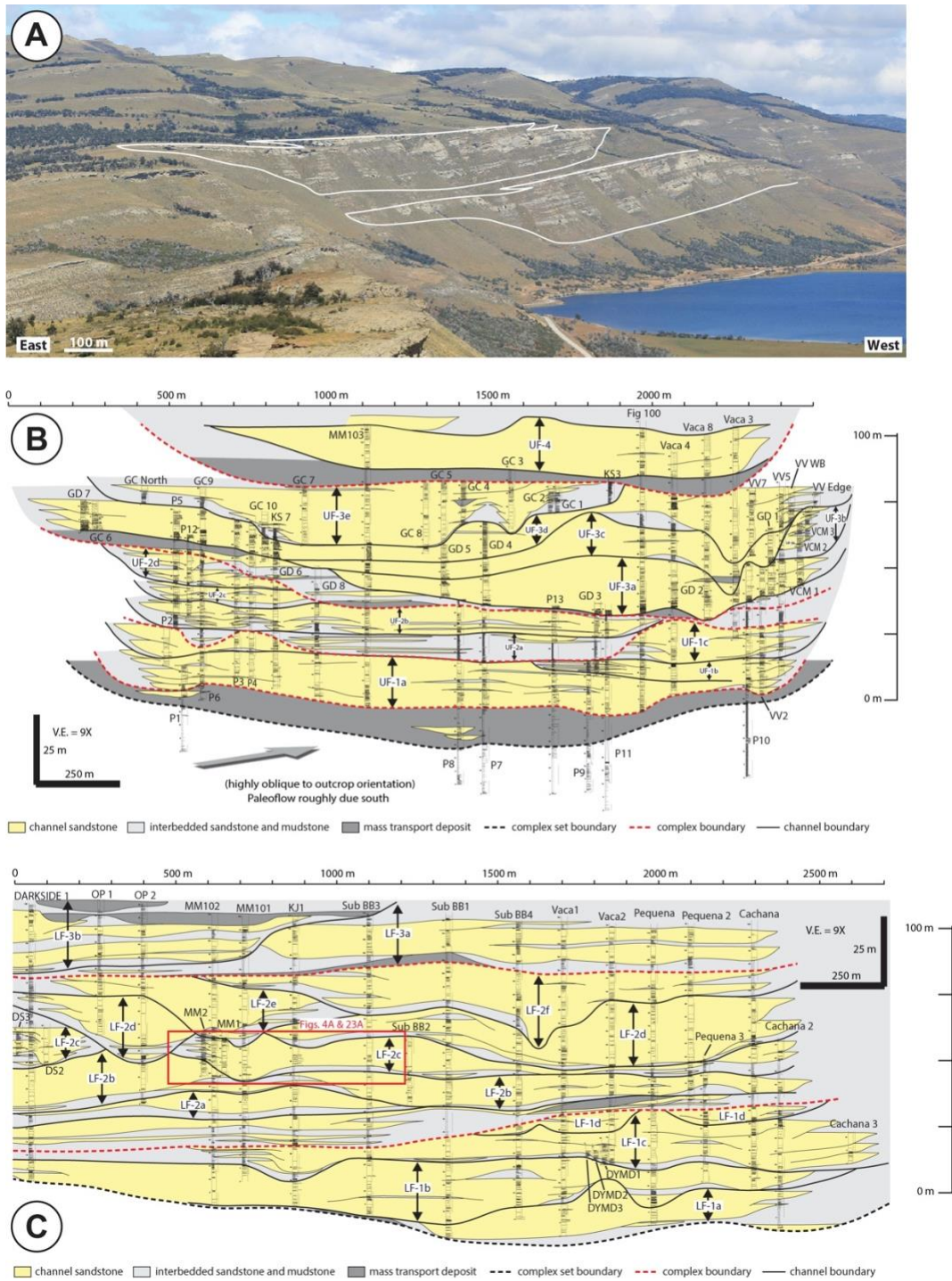


Figure 3. (A) Photo of the upper and lower channel complex sets at Laguna Figueroa with complex sets outlined (adapted from Daniels et al. (2019)). (B) Oblique dip-oriented cross section of Upper Figueroa with channel complex sets, complexes, and geobodies (4 CC; 13 CG) labeled and 41 measured sections superimposed (adapted from Southern et al., (2017)). (C) Oblique dip-oriented cross section of Lower Figueroa with channel complex sets, complexes, and geobodies labeled (3 CC; 12 CG) and 27 measured sections superimposed (adapted from Southern et al., (2017)). Red box highlights clipped image of channel geobody LF-2C shown in Figures 4A and 23A.

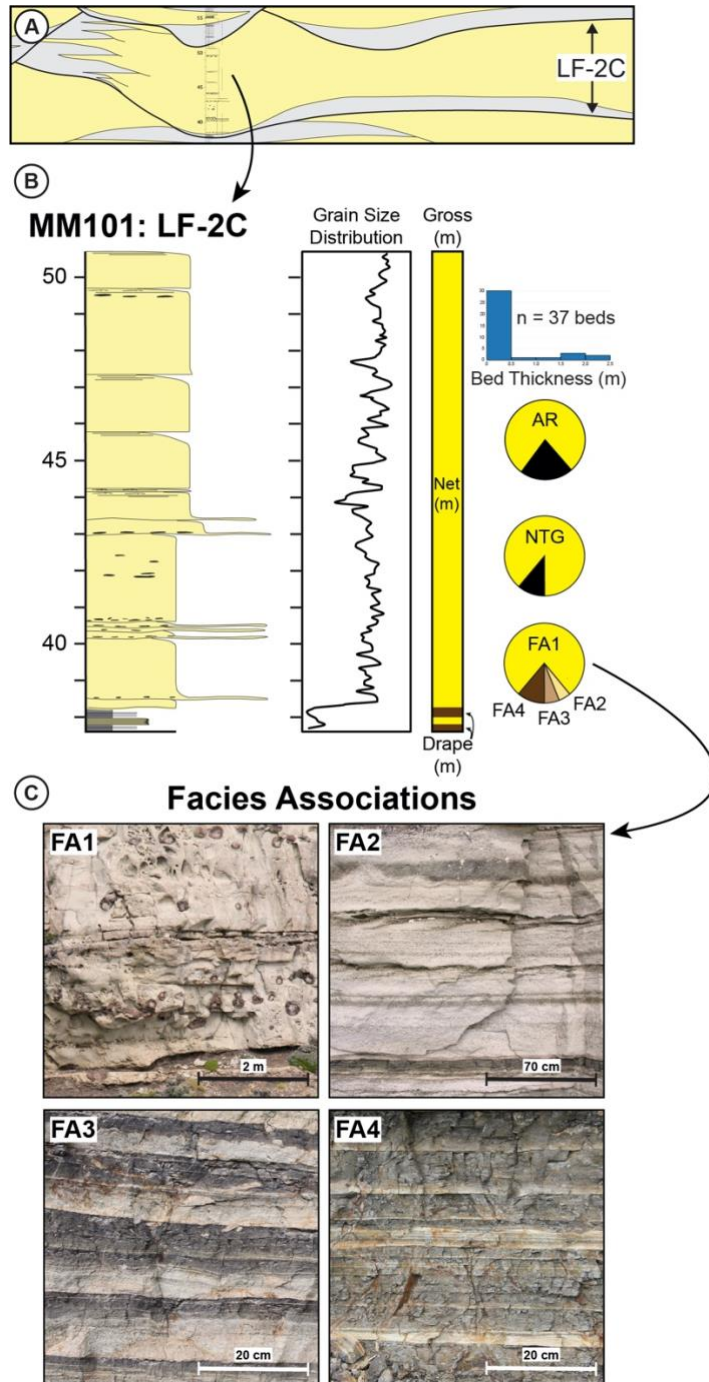


Figure 4. (A) Clipped image of channel geobody LF-2C from Figure 3C. (B) Generating the 16 features—net, gross, NTG, drape thickness, bed statistics, amalgamation ratio, and grain size distributions—for the Laguna Figueroa database from channel outcrop measured section data (MM101: LF-2C). (C) Facies associations [FA1-FA4] interpreted from Laguna Figueroa outcrops: thick-bedded, highly amalgamated sandstone (FA1); thick- to thin-bedded, semi-amalgamated sandstone and siltstone (FA2); thick- to thin-bedded, largely non-amalgamated sandstone and siltstone (sandstone-dominated) (FA3); and medium- to very-thin bedded largely non-amalgamated sandstone and siltstone (siltstone-dominated) (FA4).

2.2 Laguna Figueroa Database

The Laguna Figueroa database is sourced from stratigraphic measured section data (68 measured sections; total thickness: 3,435 m; Appendix A; Fletcher, 2013; Macauley and Hubbard, 2013; Southern et al., 2017) of exposed deep-water channel outcrops at Laguna Figueroa. The raw measured section data was originally documented in an Excel spreadsheet, which contained the names of each measured section, the channel geobodies they intersected, and various stratigraphic observations, such as number of sedimentation units, sedimentation unit thickness, amalgamation indicators, and facies associations interpretations. Additionally, for each channel geobody within each measured section, a distinct channel position—axis, off-axis, or margin—interpretation was made. This spreadsheet was imported into MATLAB to extract various channel outcrop statistics and construct the Laguna Figueroa database. The statistics extracted from the measured sections and observations of slope channel stratigraphy (Appendix A; Macauley and Hubbard, 2013; Hubbard et al., 2014) were leveraged to perform the machine learning analyses in this study.

2.2.1 Channel Outcrop Statistics

The quantitative channel outcrop statistics, referred to as features, extracted from the raw measured section data are the foundation of the Laguna Figueroa database. These features were separated into individual samples based on their respective channel geobodies which were interpreted in the field. For example, measured section MM101 intersects Lower Laguna Figueroa channel geobodies 1C, 2A, 2B, 2C, 2E, 2F, 3A, and 3B (Figure 3C). Focusing on where MM101 intersects LF-2C (Figure 4A), various features representative of the specific channel geobody can be generated (Figure 4B). In total, all of the features for LF-2C constitute one data sample.

The columns of the Laguna Figueroa database represent the individual features and the rows represent the samples or channel geobodies intersected by each measured section (Table 1).

Table 1. Example of the format for the Laguna Figueroa database (Appendix A).

Complex Set	Section Name	Geobody	Net (m)	Gross (m)	NTG	Drape Thickness (m)
Upper	FIG100	11	7.82	13.97	0.56	6.15
Upper	FIG100	10	11.84	11.84	1.00	0.00
Upper	FIG100	8	21.27	21.85	0.97	0.43
Upper	FIG100	3	7.10	10.02	0.71	0.83
Upper	FIG100	2	7.59	8.05	0.94	0.00

The entire database is 157 rows x 16 columns, meaning there are 157 samples and 16 features for each sample (Appendix A). Additionally, as previously mentioned, each sample has a channel position classification depending on its position within the channel body and the classification scheme used. Table 2 shows the number of samples per class in the classification schemes. The 16 features that comprise the database are explained below.

Table 2. Samples per class in the Facies-Driven and Geometric classification schemes.

Classification Scheme	Axis	Off-Axis	Margin
FD-2P	112	-	45
FD-3P	51	62	44
GM-2P-1	131	-	26
GM-2P-2	110	-	47
GM-3P-1	110	36	11
GM-3P-2	81	46	30

2.2.1.1 Facies Association Proportions

Facies proportions (Appendix A.1) are the proportions of each facies [FA1-FA4] within each channel body sample. Facies proportions were calculated for the database by individually summing the thicknesses of each facies within each channel body and dividing that by the overall thickness of the respective channel body.

2.2.1.2 *Net*

Net (Appendix A.1) is the total thickness of pay or sandstone within a reservoir interval. Net was calculated by summing the thicknesses of FA1-FA3 within each channel body.

2.2.1.3 *Gross*

Gross (Appendix A.1) represents the thickness of the preserved channel bodies at Laguna Figueroa. This statistic can be useful in separating axis, off-axis, and margin because channels tend to thin laterally from the thalweg to channel edge based on their channel-form shape. However, this statistic can be misleading because it does not represent true channel thickness since subsequent channels can incise and erode away the tops of previous channels.

2.2.1.4 *Net-To-Gross*

Net-to-gross (NTG; Appendix A.1) is the ratio of pay or sandstone thickness to the total thickness of the reservoir interval. NTG was calculated by taking the ratio of the net value to the gross value for each channel sample.

2.2.1.5 *Drape Thickness*

Drapes are thin mudstone deposits located along the base and/or margins of a channel body (Barton et al., 2010). These stratigraphic features represent heterogeneity between stacked channel bodies that can have significant impacts on fluid flow and reservoir connectivity. The thicknesses of drape deposits can also be important metrics for characterizing the fill character of channel elements. This statistic (Appendix A.1) was calculated from the measured section data by summing the thicknesses of beds labeled as FA4 at the base of all channel bodies.

2.2.1.6 *Bed Statistics (Count, Minimum, Median, and Maximum)*

The number of beds (Appendix A.2) is the total number of individual sedimentation units within a channel body. These beds represent the accumulation of sediment through individual

depositional events over time. Minimum, median, and maximum bed thickness (Appendix A.2) were extracted from the distribution of bed thicknesses within a channel body.

2.2.1.7 Amalgamation Ratio

Amalgamation ratio (Appendix A.2) is the ratio of the number of amalgamated surfaces—surfaces with sandstone on sandstone contact between beds—to the total number of beds minus one (Fletcher et al., 2011).

2.2.1.8 Grain Size Distributions and P10, P50, and P90 Statistics

Grain size distributions were digitized by bed at a normal sampling rate for each measured section using a MATLAB digitizer. The grain size distributions of particle diameter size in millimeters (d) were converted over to phi (ϕ) scale (Blair and McPherson, 1999) using the equation:

$$\phi = -\log_2 d$$

The P10, P50, and P90 statistics (Appendix A.3) were extracted from the converted phi scale distributions and added to the outcrop database to be used as input features for the machine learning and deep learning analyses.

2.2.2 Classification Schemes

In this study, several different classification schemes representing alternate methods for classifying channel position (axis, off-axis and margin) were used to test and compare the efficacy of using one scheme over another. Sedimentological groupings are not straightforward nor immediately apparent, and in practice, sedimentologists group based on similar properties. These designations are highly subjective, which motivates the need to explore the impact of different schemes. The different schemes implemented in this study can be separated into two major categories, facies-driven (FD) and geometric (GM), with subcategories based on the number of

desired classifications—two positions or three positions. The goal behind using different schemes is to assess the difference 1) between subjective/biased (FD) and non-subjective/unbiased classification (GM) schemes, and 2) less versus more classes (i.e., lumping vs. splitting). These schemes are described in greater detail below.

2.2.2.1 Facies-Driven

The FD classification scheme (Appendix A.4) was generated from the field data (Macauley and Hubbard, 2013) based on observations of sedimentary structures, facies proportions, and other distinguishing characteristics of the Laguna Figueroa outcrops. The subcategories of the FD scheme include a two-position scheme (FD-2P) and a three-position scheme (FD-3P). The FD-2P scheme (Figure 5) classifies the sets of channel outcrop features into either axis or margin. This scheme does not account for an off-axis classification, which can often be difficult to distinguish in the field due to the transitional nature of the position within a channel body. Conversely, the FD-3P scheme (Figure 5) includes this transitional zone and classifies the data into axis, off-axis, or margin. The facies-driven scheme is based solely upon groupings from visual differentiation and observation based on geologic intuition. It is suspected that this method oversimplifies and biases grouping because it is based on visual similarities in facies.

2.2.2.2 Geometric

Since the FD classification schemes are interpretive and based on human decision making, which can be prone to error, the need for an objective classification scheme was recognized and thus the GM classification scheme was born. The GM classification scheme (Appendix A.4) separates channel bodies into different positions based on height above the bottom of a channel body. The subcategories for the GM classification scheme include two different two-position schemes (axis or margin) and two different three-position schemes (axis, off-axis, or margin).

The first two-position scheme (GM-2P-1) classifies channel bodies into axis or margin based on a height of 12 meters above the bottom of a channel body (Figure 5). This cutoff was selected to split the average channel body thickness for Laguna Figueroa, 24 meters, into halves. The second two-position scheme (GM-2P-2) uses a cutoff of 8 m above the bottom of a channel body (Figure 5). This cutoff was reduced from the initial 12 m cutoff to try to capture the general difference between amalgamated and sand-prone channel fills versus more heterolithic channel fills.

The first three-position classification scheme (GM-3P-1) classifies channel bodies into axis, off-axis, or margin using cutoffs of 8 m and 16 m above the bottom of a channel body (Figure 5). These cutoffs were selected by splitting the average channel body thickness for Laguna Figueroa into thirds. Although it is objective, this particular classification scheme does not capture the margin position well for the available data. Therefore, the second three-position classification scheme (GM-3P-2) was created by reducing the cutoffs from 8 meters and 16 meters to 5 meters and 11 meters to try to capture the general variation in channel position throughout the outcrops (Figure 5).

2.2.3 Generation of Hypotheses

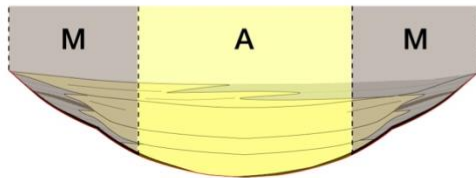
Once the features for the database were generated and labeled accordingly for the facies-driven and geometric schemes, the data was analyzed using density plots—kernel-smoothed histograms—to form hypotheses for the machine learning analyses. The density plots for the FD-2P (Figure 6A) and FD-3P (Figure 6B) schemes show that channel geobody axis positions are generally well-defined and characterized by higher proportions of FA1 and lower proportions of FA4, larger gross thicknesses, higher NTG, thinner drapes, fewer and thicker beds, and higher amalgamation ratios. Contrarily, margins constitute the siltstone-dominated and thinner end-

members of the channel position spectrum with higher proportions of FA4, smaller gross thicknesses, lower NTG, thicker drapes, numerous thin beds, and low amalgamation ratios (Figure 6B). The off-axis position is transitional between these two end members with greater spreads in the distributions of its features. These general trends and defining characteristics for each channel position are also exemplified by the geometric schemes for both two positions (Figure 7) and three positions (Figure 8). However, there is much more overlap between the features in the geometric schemes since it is not influenced by geologic interpretation. Additionally, this overlap appears to be intensified in the schemes with the smaller cutoffs from the base of a channel geobody for channel position (i.e., GM-2P-2 and GM-3P-2; Figures 7B and 8B).

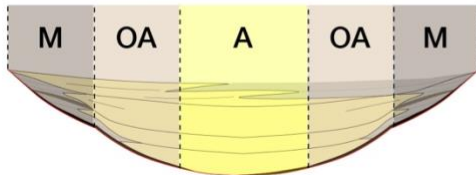
Several hypotheses were formed based on these observations in anticipation of the results of the machine learning analysis executed in this study. Firstly, since the different channel positions are relatively defined by the different classification schemes, it was hypothesized that the machine learning algorithms would be useful for predicting channel position from the Laguna Figueroa database. Secondly, because off-axis is less defined and more transitional as a channel position, it was hypothesized that the machine learning algorithms would struggle to predict this position versus the others. Lastly, the facies-driven classification schemes separate the different channel positions the best. Consequently, it was hypothesized that the algorithms would be more successful at classifying data for the facies-driven schemes than the geometric scheme counterparts. These hypotheses were tested using a range of machine learning algorithms and analyzing their performance for each classification scheme and channel position.

FACIES-DRIVEN

Two Positions (FD-2P)

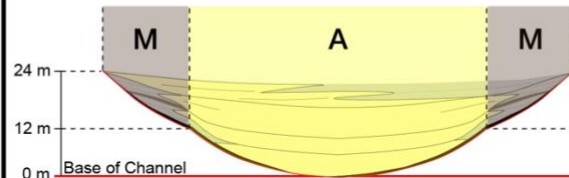


Three Positions (FD-3P)

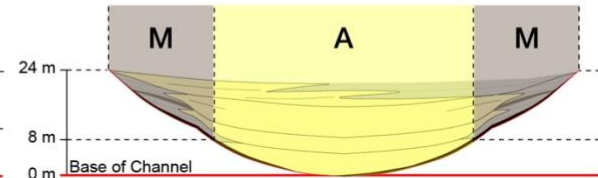


GEOMETRIC

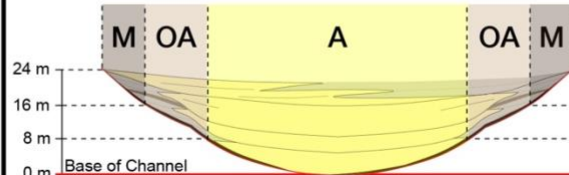
Two Positions (GM-2P-1)



Two Positions (GM-2P-2)



Three Positions (GM-3P-1)



Three Positions (GM-3P-2)

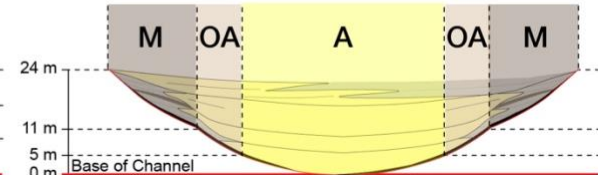


Figure 5. Channel position classification schemes for outcrop data from Laguna Figueroa separated into two categories—facies-driven (FD) and geometric (GM)—with varying numbers of channel positions—axis (A), off-axis (OA), or margin (M)—for each scheme. The FD classification scheme is based on observations and interpretations of intra-channel facies made in the field and can be separated into either two positions (FD-2P), or three positions (FD-3P). The GM classification scheme is an attempt at an objective classification scheme that doesn't rely on interpretation. This scheme is based on height above the base of a channel and can be separated into two different two-position schemes (GM-2P-1 or GM-2P-2) or two different three-position schemes (GM-3P-1 or GM-3P-2). The GM-2P-1 scheme uses a cutoff of 12 m and the GM-2P-2 scheme uses a cutoff of 8 m. The GM-3P-1 scheme uses cutoffs of 8 m and 16 m and the GM-3P-2 scheme uses cutoffs of 5 m and 11 m.

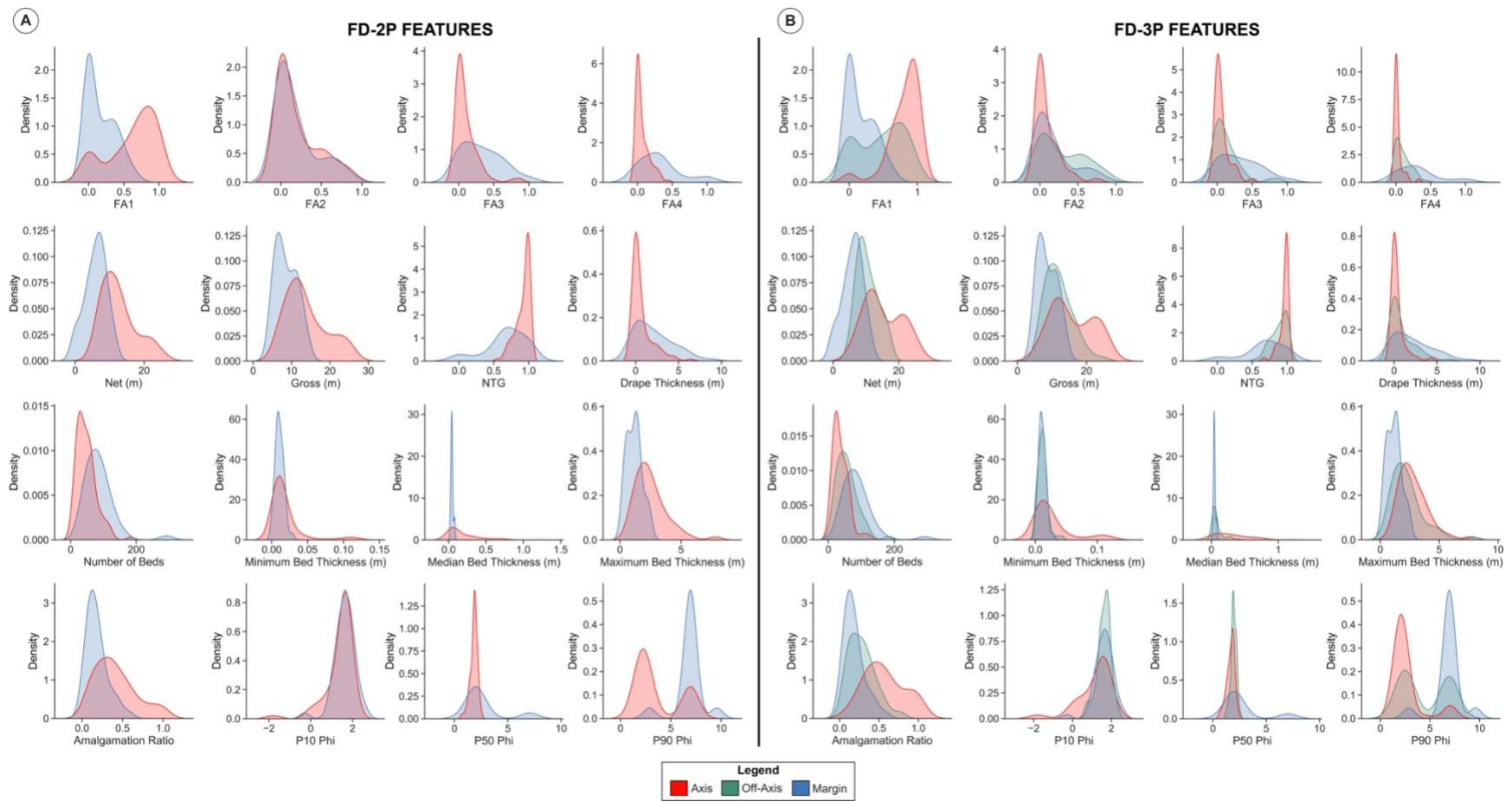


Figure 6. Facies-driven 2P (A) and 3P (B) feature distributions for axis, off-axis, and margin data from Laguna Figueroa database.

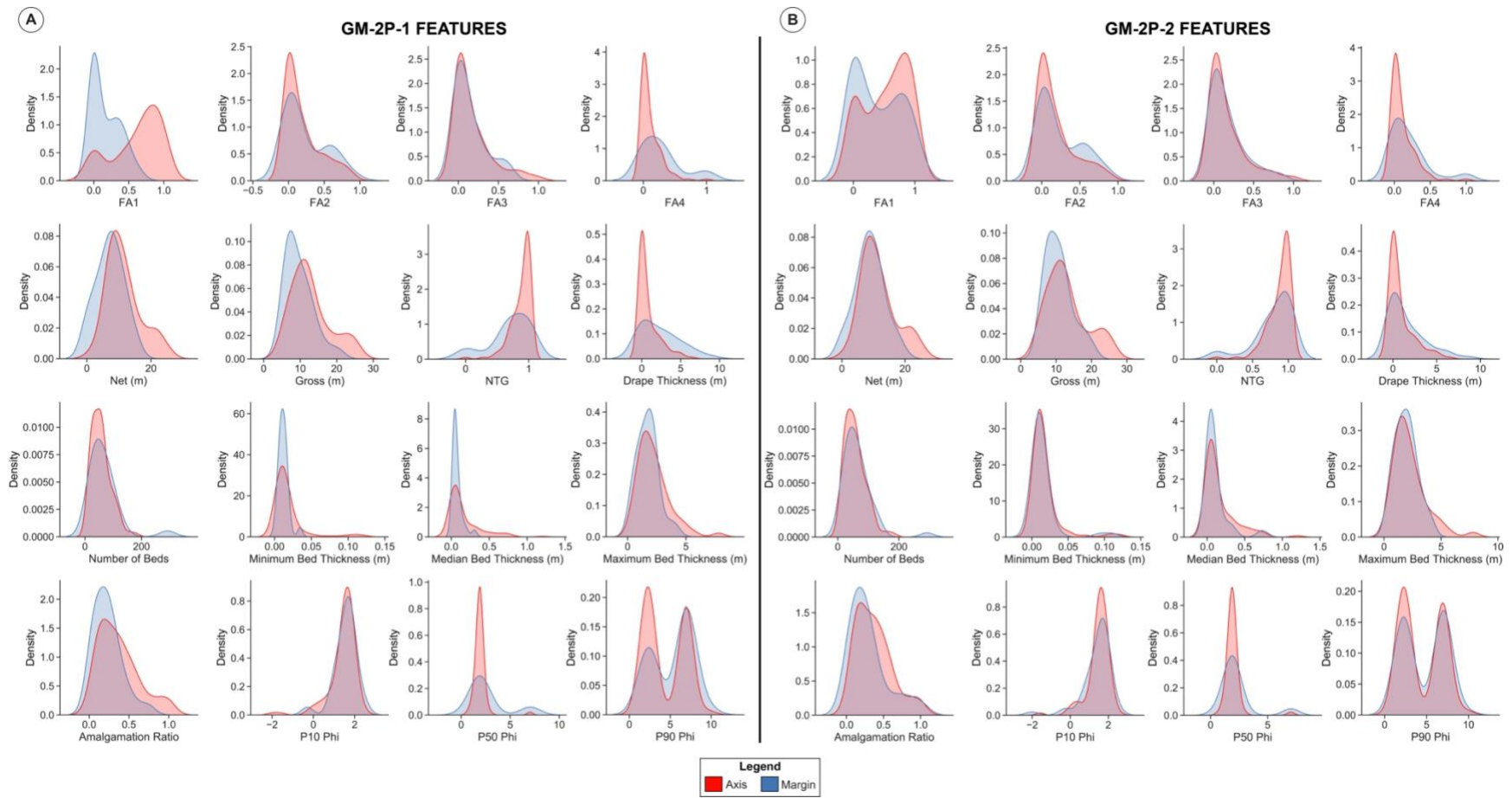


Figure 7. Geometric 2P-1 (A) and 2P-2 (B) feature distributions for axis and margin data from Laguna Figueroa database.

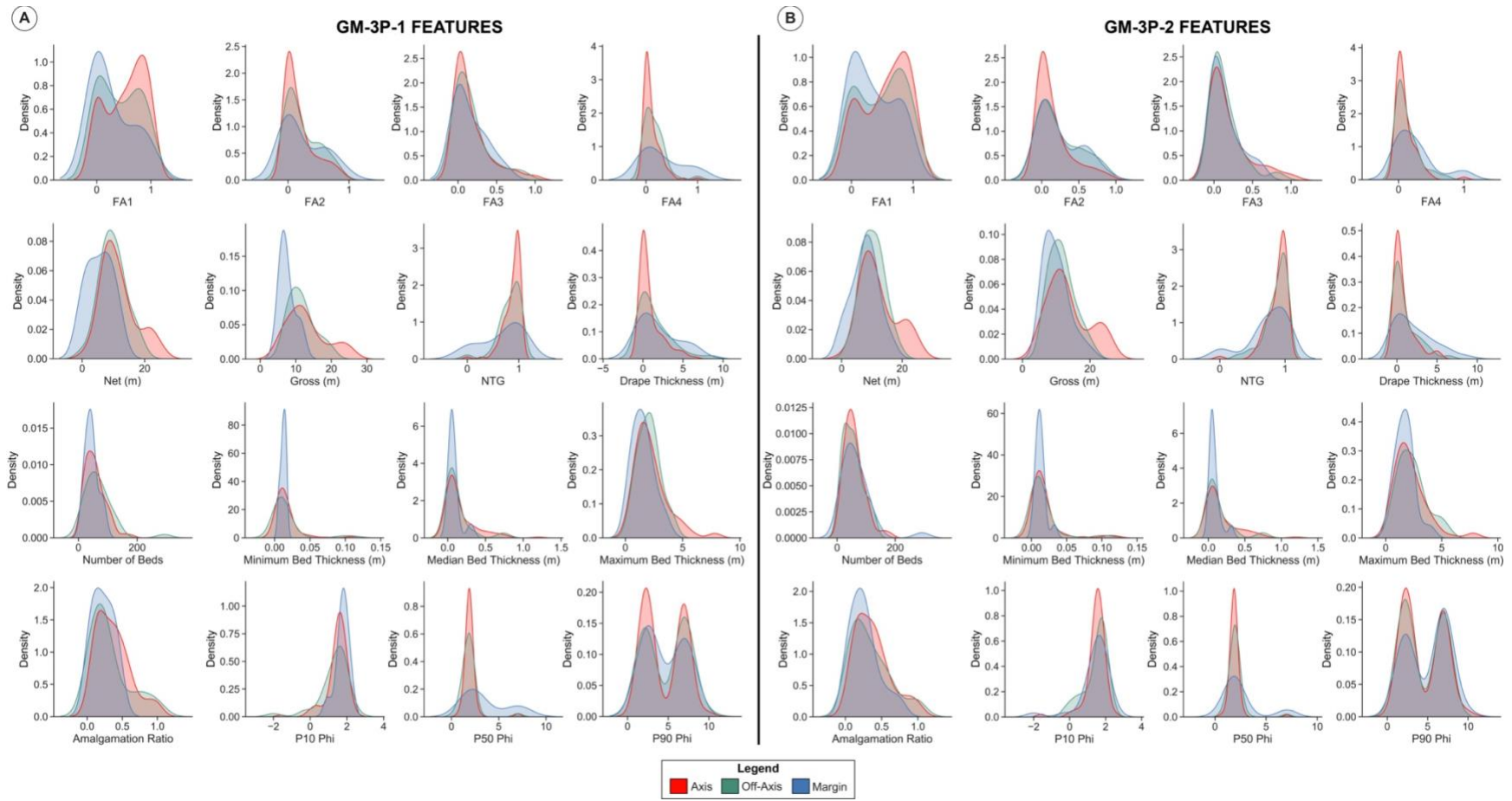


Figure 8. Geometric 3P-1 (A) and 3P-2 (B) distributions for axis, off-axis, and margin data from Laguna Figueroa database.

CHAPTER 3: MACHINE LEARNING OVERVIEW

3.1 Machine Learning Algorithms

In this study, a range of different machine learning algorithms—unsupervised, supervised, and deep learning—are used to analyze and interpret the channel outcrop statistics from the Laguna Figueroa database. These algorithms are described in further detail below.

3.1.1 Unsupervised Learning

3.1.1.1 Feature Importance with Principal Component Analysis

Principal component analysis (PCA) is a multivariate statistical technique used to analyze datasets containing inter-correlated quantitative dependent variables (Abdi and Williams, 2010). It is useful for determining feature importance and/or reducing the dimensionality of these datasets by identifying the directions of maximal variance within the data, otherwise known as principal components (PCs; Figure 9A).

The goal of PCA is to find the best representation of the data using a finite number of PCs (Lever et al., 2017). The directions of the PCs in feature space are represented as eigenvectors while their magnitudes are represented as eigenvalues. PCA assumes that the largest eigenvalues retain the most important information (i.e. largest variances) in the data (Wold et al., 1987). In addition to eigenvalues, features can be analyzed further based on their contributions. Contributions represent how important a feature is for a principal component. Features that have higher contributions for the last dimensions are less important than those that contribute more to the primary PCs (Kassambara, 2017).

In this study, a method for PCA in R (Kassambara, 2017) was used to determine feature importance for all 157 samples and 16 features from the Laguna Figueroa dataset.

3.1.1.2 Clustering Analysis with K-Means

K-means is an unsupervised learning algorithm initially proposed by (MacQueen, 1967) that partitions n -dimensional data into k clusters based on each data point's distance from the mean of the nearest cluster. The algorithm works similarly to a gradient descent algorithm in which cluster centroids are generated randomly and then updated after each iteration until convergence is reached (Oyelade et al., 2010; Figure 9B). In this study, K-means was used to cluster the Laguna Figueroa data without any human bias into statistically important zones, ideally representative of the different channel position classification schemes. Since K-means is an unsupervised algorithm and doesn't know what the correct classifications are for each data sample, the clusters generated by the clustering analysis were manually interpreted as different channel positions to extract value from the analysis and evaluate the performance of the K-means algorithm.

3.1.2 Supervised Learning

In this study, 26 supervised learning algorithms were used to classify channel position from the Laguna Figueroa database. These algorithms range from variations of decision trees, discriminant analysis, Naïve Bayes, support vector machines, K-nearest neighbors, ensemble classifiers, and neural networks (Figure 9).

3.1.2.1 Decision Trees

Decision trees are algorithms that work similarly to flowcharts, in which various root nodes, internal nodes, leaf nodes, and branches denoting various decision points and decisions separate input data into different classes (Figure 9C). Root nodes represent an initial decision point in which all samples will be divided into two or more subsets (Song and Lu, 2015). All input samples enter the decision tree through the root node. Once initial subsets are created, they encounter internal nodes which filter the data further. If there are no more internal nodes in the

tree, and the data cannot be filtered further, they reach leaf nodes, which correspond to the final classifications. All of the nodes are connected by branches, which represent decisions made at the root and internal nodes. For example, if the root node signifies a decision of whether the input data is greater than or equal to 0.5, then the branches extending from the root node would signify the two possible outcomes: yes, the data is greater than or equal to 0.5; or no, the data is less than 0.5. Decisions trees can either be fine, medium, or coarse, depending on the number of nodes that they are composed of (Han et al., 2019). In this analysis, all three deviations are tested.

3.1.2.2 Discriminant Analysis

Discriminant analysis is a classification algorithm that assumes samples from each class form multidimensional Gaussian distributions (Dixon and Brereton, 2009; Figure 9D). The algorithm is first trained to fit a function to estimate the distribution parameters of each class, and then it is tested by predicting new samples (Welling, 2005; Han et al., 2019). Two common forms of discriminant analysis exist: linear discriminant analysis (LDA) and quadratic discriminant analysis (QDA). In LDA, classes are assumed to have the same variance-covariance matrices or distributions, which results in linear boundaries or hyperplanes separating the feature space into classes (Tang et al., 2017; Dixon and Brereton, 2009). Whereas in QDA, different classes are assumed to have unique variance-covariance matrices, which results in quadratic curves that divide the feature space into classes. Both LDA and QDA are tested in this study.

3.1.2.3 Naïve Bayes

Naïve Bayes is a type of probabilistic classifier that utilizes Bayes Theorem and assumes all features, x , are independent given the class variable, c (Zhang, 2005; Figure 9E):

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

where $P(c|x)$ is the probability of the class given the feature; $P(x|c)$ is the probability of the feature given the class; $P(c)$ is the prior probability of the class; and $P(x)$ is the prior probability of the feature. Other forms of the Naïve Bayes classifier include Gaussian and Kernel Naïve Bayes, which assume specific distributions for the input data.

3.1.2.4 Support Vector Machines

Support vector machines (SVMs) separate the data into different classes using hyperplanes that maximize the distance between classes (Tien Bui et al., 2012; Figure 9F). Different kernel functions can be applied to SVMs to determine the boundaries between classes. These kernel functions include linear, quadratic, cubic, and Gaussian. Additionally, the flexibility of the Gaussian SVM can be adjusted to fine-, medium-, or coarse-scale depending on the desired distinctions between classes. All six of these SVMs were tested in this study.

3.1.2.5 K-Nearest Neighbors

K-nearest neighbors (KNN) is one of the simplest supervised learning algorithms available for classification. KNN works by classifying new testing data based on the dominant class of its k nearest neighbors in feature space (Figure 9G). The nearest neighbors are typically determined using Euclidean Distance:

$$d(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

where $d(x,y)$ is the distance between samples x and y , i is the current feature, and n is the total number of features (Ozcoban et al., 2018). Depending on the value specified for k , the KNN algorithm can either be fine, medium, or coarse. Additionally, other variations of KNN exist which use different distance metrics, such as cosine KNN, cubic KNN, and weighted KNN. In this study, all six KNN algorithms were used.

3.1.2.6 Ensemble Classifiers

Ensemble classifiers are models that combine the prediction abilities of multiple machine learning algorithms in order to improve the performance of a model or reduce the potential of selecting a bad model (Polikar, 2012). In ensemble learning, weaker learning algorithms, such as decision trees and KNN, are improved using techniques, such as bagging, random subspace method (RSM), and boosting (Abuassba et al., 2017; Han et al., 2019). Weak learners are algorithms that are slightly better than random guessing (Freund and Schapire, 1997).

Bagging, also known as bootstrap aggregating, takes the original training data, draws samples with replacement, and generates new training data subsets which are then used to train different classifiers (Polikar, 2012; Brieman, 1994). The ensemble classifier then makes a classification by taking a majority vote of the predictions made by all of the classifiers (Figure 9H). For example, random forest (RF) is a popular ensemble classifier that utilizes multiple decision trees and bagging to generate classifications by committee rather than from a single decision tree. The idea behind this is strength in numbers—some trees will be wrong, but ideally, most will be right, and therefore a correct classification will be made. RSM is similar to bagging, but instead of bootstrapping the training data, RSM bootstraps in the feature space (Skurichina and Duin, 2002; Tao et al., 2006; Figure 9I). In other words, RSM constructs classifiers based on pseudo-random components of the input features (Kotsiantis 2011).

In boosting, weak classifiers are generated sequentially. Each new classifier is trained on a weighted dataset from the previous classifier thus resulting in improved classifications (Friedman et al., 2000; Figure 9J). Each iteration produces three different classifiers: C_1 , which is trained with a random subset from the training data; C_2 , which is trained on a subset of data that is an even mix of data correctly classified by C_1 and data incorrectly classified by C_1 ; and C_3 , which is trained on

data the classifiers C_1 and C_2 disagree on (Polikar, 2012). Like bagging, the output of the boosting model is a majority vote decision of all of the classifiers (Freund and Schapire, 1996).

Similar to RF being a combination of decision trees and bagging, other ensemble classifiers, such as AdaBoost, RUSBoosted trees, and XGBoost (XGB), combine decision trees with boosting methods to improve overall performance. Adaptive boosting, or AdaBoost, works by training a specified number of weak classifiers on weighted versions of the training data samples with the misclassified samples being weighted the highest (Friedman et al., 2000; Wyner et al., 2017). This process is repeated for a sequence of weighted samples, and the final output is a linear combination of all of the classifiers (Friedman et al., 2000; Wyner et al., 2017).

Random undersampling boosted trees, or RUSBoosted trees, are a type of ensemble-of-trees classifiers that are applied to skewed datasets—datasets in which samples of one class severely outnumber samples of the other class(es) (Seiffert et al., 2010). RUS is a resampling method in which samples of the majority class are stochastically removed until the dataset is balanced (Seiffert et al., 2010). The RUSBoosted algorithm combines this data-balancing approach with the powers of AdaBoost to improve classification results.

Extreme gradient boosting, or XGBoost, is a scalable tree boosting system that was first introduced by Chen and Guestrin (2016). The system is based on gradient boosting, which is another ensemble method similar to AdaBoost. However, instead of applying higher weights to misclassified samples, gradient boosting applies a gradient descent algorithm (Qian 1999) to minimize the error between the weak classifier’s predictions and the observed values (i.e., residuals) from the training dataset (Friedman 2002). Subsequent classifiers are then constructed based on the minimized residuals of the previous classifiers, and the process is repeated. The boosted trees in XGBoost benefit from innovations such as scalability, a regularization parameter

for the loss function that avoids over-fitting, ability to handle sparse data, and novel weight-adjustment methods (Chen and Guestrin, 2016). These improvements to the gradient boosting framework have allowed XGBoost to be the winning algorithm of many machine learning challenges on the site Kaggle (Chen and Guestrin, 2016).

In this study, Ensemble Classifiers: Random Forest, Subspace Discriminant, Subspace KNN, AdaBoost Trees, RUSBoosted Trees, and XGBoost were tested.

3.1.2.7 K-Fold Cross-Validation

K-fold cross-validation is a resampling technique used in machine learning analysis in which the input dataset is split into k folds with $k-1$ folds being used for training and the remaining fold being used for testing (Figure 10). This process is repeated for each possible combination of the $k-1$ training folds and k testing folds, so that each of the k folds is used as a testing fold. The evaluation metrics are then averaged for all of the k testing folds in order to obtain a more representative evaluation of the machine learning algorithm. Five folds were used in the machine learning analyses in this study.

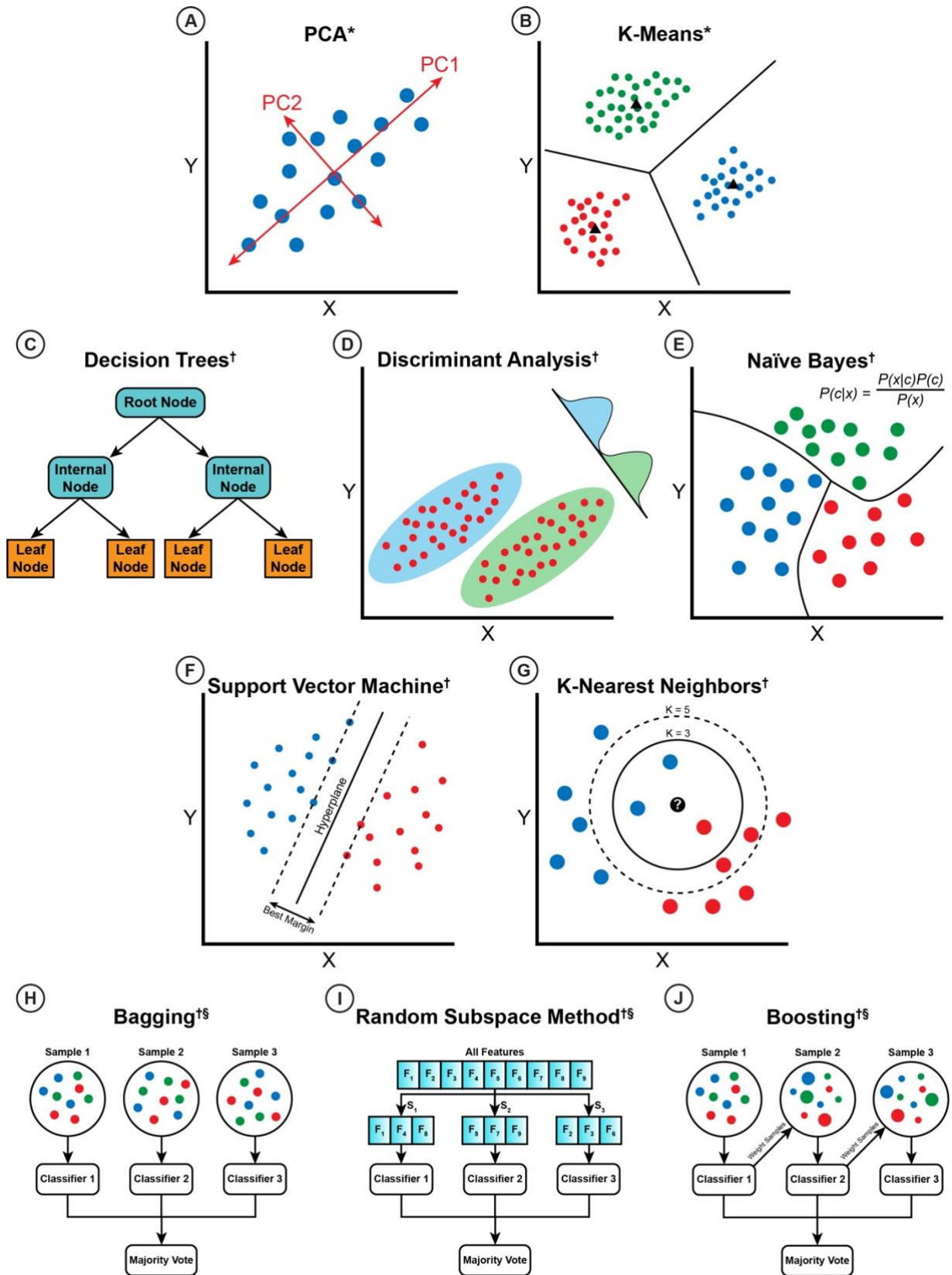


Figure 9. Unsupervised*, supervised†, and ensemble§ learning algorithms.



Figure 10. K-fold ($k = 5$) cross-validation. All data samples are split into five different folds. In each fold, the machine learning algorithm is trained on four of the folds and is tested on the remaining fold. The overall performance of the algorithm is calculated by averaging the results of the test folds.

3.1.3 Deep Learning

3.1.3.1 Neural Networks

Artificial neural networks (ANNs) are algorithms loosely modeled after biological nervous systems (McCulloch and Pitts, 1943). They consist of multiple layers—an input layer, hidden layer(s), and an output layer—each made up of interconnected units called neurons (Rosenblatt, 1958; Rosenblatt, 1962). The input data enter the ANN through the neurons in the input layer and are subsequently fed forward to neurons in subsequent layers by individual weights and biases that connect the layers. The values of these weights and biases cause specific neurons in the subsequent layers to activate, which ultimately results in a classification in the output layer. Through numerous iterations or epochs, the weights and biases are updated to improve the classification accuracy of the ANN. This process is called backpropagation (Rumelhart et al., 1986; Lecun et al., 1989) and it is made possible by optimization algorithms, such as gradient descent (Qian, 1999), stochastic gradient descent (Gardner, 1984), and conjugate gradient methods (Shewchuk, 1994), which seek

to minimize a specified loss function. Loss functions represent the error between the ANNs predictions and the observed values. Common loss functions include mean square error, mean absolute error, and cross-entropy loss.

ANNs have been useful for many data science problems due to their ability to learn non-linear functions and thus detect complex relationships within datasets (Kawabata and Bandibas, 2009; Goodfellow et al., 2016). The ANN architecture applied in this study consisted of an input layer with sixteen neurons, $L_1^{[16]}$, a hidden layer with five neurons, $L_2^{[5]}$, and an output layer with k neurons corresponding to the number of desired classifications, $L_3^{[k]}$ (Figure 11). Cross-entropy loss was used as the loss function for the ANN:

$$CE = - \sum_{i=1}^n t_i \log(f(s)_i)$$

A scaled conjugate gradient (SCG) algorithm (Appendix B; Møller, 1993; Anderson et al., 2011) was used for optimization and backpropagation. The SCG algorithm combines the advantage of other conjugate gradient methods by searching in the conjugate directions of the derivative of the objective function while also approximating a localized quadratic function (Møller, 1993). A softmax function was implemented to compute the probabilities from the ANN classifications by taking the logits—the raw values of the output layer—and transforming them into a multinomial distribution:

$$softmax(x_i) = \frac{\exp(x_i - x_{i,max})}{\sum_{j=1}^n \exp(x_j)}$$

The probabilities from the softmax function can be used to build stochastic models of the most probable channel stacking scenarios at a wellbore. The different models can be used to visualize the impact of channel stacking patterns on fluid flow and connectivity.

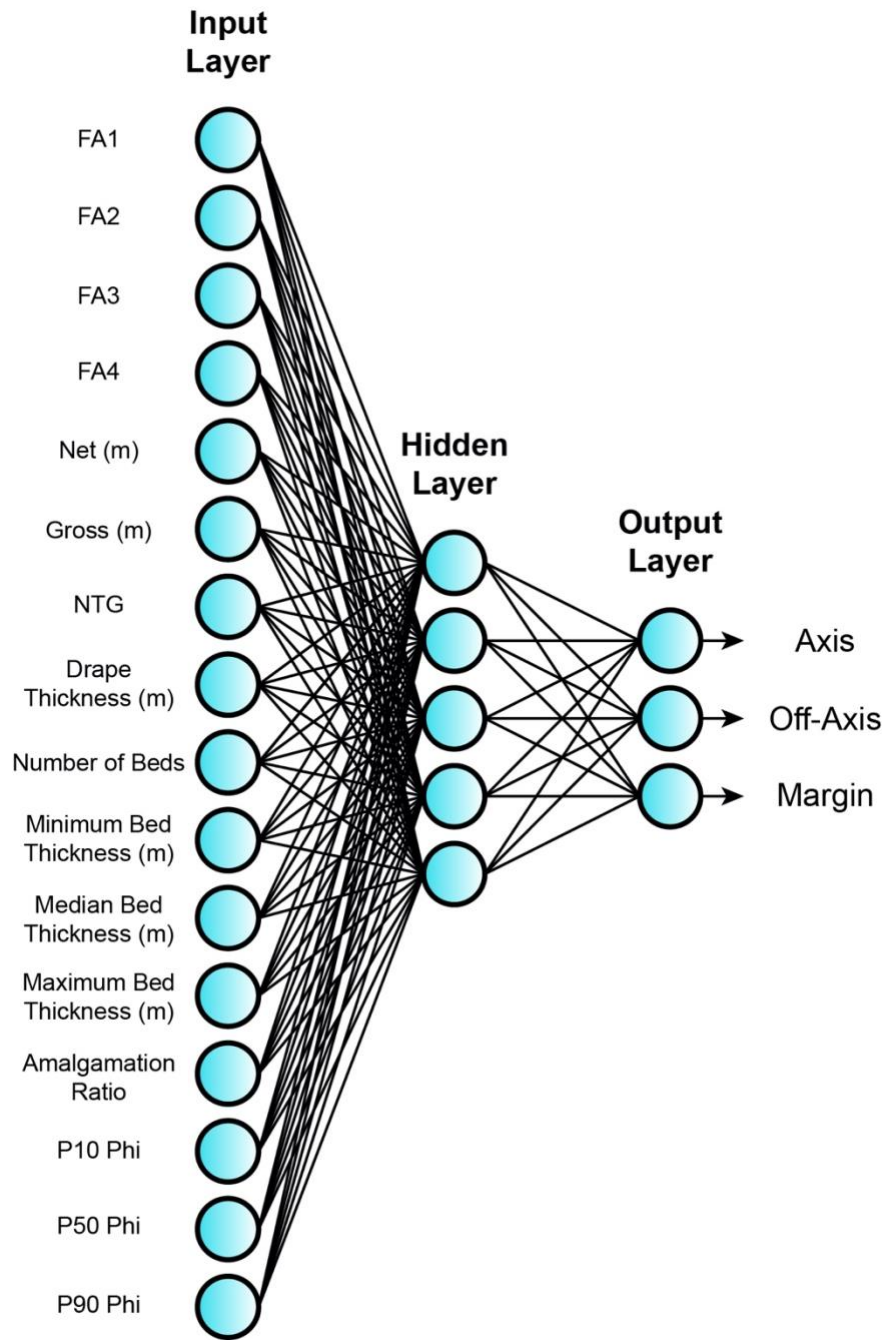


Figure 11. ANN architecture for classifying a three-position scheme. Measured section data (Hubbard et al., 2018) is converted into outcrop statistics (Figure 3) that are used as inputs for the ANN. The ANN architecture consists of an input layer with 16 neurons representative of the 16 input features for the analysis, a hidden layer with 5 neurons, and an output layer with 3 neurons corresponding to axis, off-axis, and margin.

3.2 Evaluation Metrics

Evaluation metrics are statistics used to assess the performance of each machine learning algorithm. The evaluation metrics applied in this study include classification accuracy, confusion matrices, precision, recall, and F1 score.

3.2.1 Validation Accuracy

Validation accuracy is the ratio between the number of correct classifications and the total number of input samples for the testing dataset. An effective machine learning algorithm will have a high validation accuracy.

$$\text{Validation Accuracy} = \frac{\text{Number of Correct Classifications}}{\text{Total Number of Input Samples}}$$

3.2.2 Confusion Matrix

A confusion matrix is an $n \times n$ matrix that illustrates the performance of an algorithm, where n is the number of classes. The rows in the matrix represent the true samples for each class and the columns represent the predicted samples for each class. A perfect classification model will produce a confusion matrix with the number of predicted observations equaling the number of true observations for each class.

3.2.3 Precision

Precision is the ratio of true positives to total positives. A true positive is a predicted classification that matches the true classification. The total positives statistic is the sum of the true positives and false positives. A false positive is a predicted classification that incorrectly predicts the positive class. In other words, precision can be viewed as the likelihood that a classified sample belongs to the class that it is predicted as (Hall, 2016).

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

3.2.4 Recall

Recall is the ratio of true positives to the total number of samples that should have been classified as the positive class. The latter is the sum of the true positives and the false negatives. A false negative is a predicted classification that incorrectly predicts the negative class. Recall is often referred to as sensitivity and can be interpreted as the likelihood that a sample will be correctly classified for a particular class (Hall, 2016).

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

3.2.5 F1 Score

The F1 score is the harmonic mean of both the precision and recall. It assists in evaluating the accuracy of the algorithm, thus the higher the F1 score is, the better the algorithm is performing.

$$F1 = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$$

CHAPTER 4: EVALUATING MACHINE LEARNING ALGORITHMS FOR PREDICTION OF CHANNEL POSITION

4.1 Methodology

In this part of the study, each of the machine learning algorithms were used to analyze the Laguna Figueroa database. First, the features of the database were ranked using PCA to provide insight into feature importance. Second, K-means was used to gain insight into how the data clustered within each of the classification schemes. Third, the supervised learning algorithms were trained and tested for predicting channel position from the different classification schemes. Lastly, once all the algorithms had analyzed the data, the evaluation metrics for each algorithm were compared to test the hypotheses previously mentioned and draw conclusions.

4.2 Results

4.2.1 Unsupervised Learning Results

4.2.1.1 Feature Importance with Principal Component Analysis

Eigenvalues were calculated for all of the features in the dataset (Table 3) to determine which features were the most important. Table 3 shows that the first five PCs account for 81.6% of the variance in the data. Therefore, only these PCs were considered for analyzing feature importance.

Table 3. Eigenvalues, variance percent, and cumulative variance percent for the 16 principal components.

Dimension	Eigenvalue	Variance Percent	Cum. Variance Percent
Dim. 1	6.33	39.55	39.55
Dim. 2	2.41	15.06	54.61
Dim. 3	2.32	14.48	69.09
Dim. 4	1.19	7.43	76.52
Dim. 5	0.81	5.05	81.57
Dim. 6	0.75	4.67	86.24
Dim. 7	0.61	3.81	90.05
Dim. 8	0.43	2.69	92.74
Dim. 9	0.37	2.30	95.04
Dim. 10	0.28	1.76	96.80
Dim. 11	0.21	1.31	98.10
Dim. 12	0.17	1.03	99.14
Dim. 13	0.11	0.69	99.82
Dim. 14	0.02	0.15	99.97
Dim. 15	0.00	0.02	99.99
Dim. 16	0.00	0.01	100.00

The bubble plot of contributions (Figure 12) shows that minimum and maximum bed thickness, P10 phi, gross, FA2, and FA3 are the least important features for the first PC. Firstly, bed thickness within a channel geobody varies based on position. Axes are characterized by thicker, sandstone beds from higher energy deposits while margins are characterized by thinner, finer-grained siltstone beds from lower energy deposits (Southern et al., 2017). However, due to heterogeneity and differences in intra-channel fill and architecture, maximum bed thickness is not necessarily indicative of channel position. For example, if thick sandstone beds in a channel axis extend laterally into a sandier channel margin, then maximum bed thickness would be similar for both channel positions and thus confounding for prediction. Conversely, if a channel geobody were to have a thick, laterally extensive drape at its base with numerous thin beds, then minimum bed thickness would be similar across channel positions. Due to these scenarios, median bed thickness

and the number of beds are better metrics to use as they account for both thinner and thicker beds within each channel position.

Secondly, channels are commonly characterized by having a U-form shape with the thickest portions centered around the axis of the channel and the thinnest portions occurring near the margins of the channel. Gross channel thickness of a channel body at the outcrop is representative of preserved thickness and is not representative of the true geobody thickness as a feature. Deep-water channel outcrops represent the spatial and temporal variations in channel stacking patterns and evolution over time. Avulsion and other processes cause channels to cut and incise previous channel deposits (Zhang et al., 2017; Lowe et al., 2019), leaving behind remnants of the original channel shape and size. This results in gross thickness values that are highly variable and less representative of the true geobody thickness that might describe a channel position (i.e., thinner at the margin and thicker at the axis).

Lastly, FA2 and FA3 proportions represent the intermediate facies associations between the highly amalgamated sandstone facies association, FA1, and the siltstone-dominated facies association, FA4. FA1 and FA4 are highly distinguishing of axis and margin channel positions since axes are typically characterized by thicker, amalgamated sandstones, and margins are characterized by higher proportions of siltstone and mudstone. FA2 and FA3 are more transitional facies associations that lack the ability to clearly differentiate channel position.

These results provide insight into what features to consider when attempting to interpret axis, off-axis and margin from core data. Specifically, the four most important features (FA4, NTG, AR and FA1) all clearly differentiate axis from margin, where margin has a high proportion of FA4, low NTG, AR and proportion of FA1 and axis has a high proportion of FA1, high NTG and AR, and a low proportion of FA4. The impact of using a reduced set of features was tested in

the machine learning analysis to ascertain if using all 16 features added noise or if all 16 features played an important role in prediction. The results revealed higher prediction accuracies when all 16 features were used, and therefore, the additional features carry information, albeit weaker information, about channel position. Therefore, the following analyses use all 16 features from the Laguna Figueroa database.

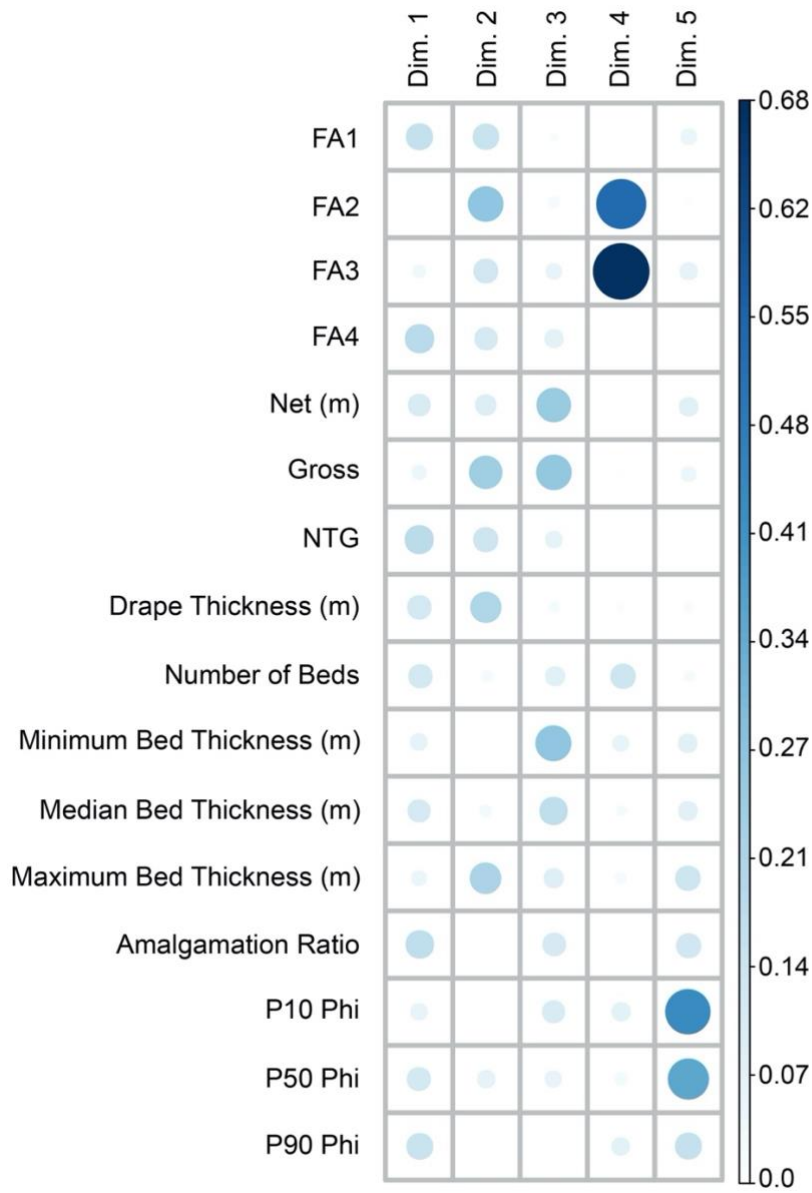


Figure 12. PCA results showing feature importance based on contribution to the first five PCs for the 16 features from the Laguna Figueroa database.

4.2.1.2 Clustering Analysis with K-Means

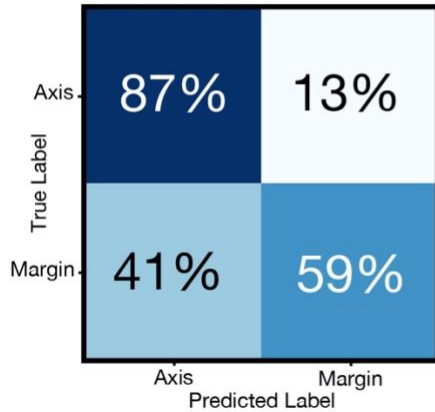
K-means was used to cluster the 16-feature dataset described above into two and three clusters, representative of the number of channel positions in the different classification schemes. The classification accuracies were calculated by comparing the K-means predictions for k clusters to the classification schemes with the same number of channel positions (Table 4). The results show that for the FD and GM schemes, FD-2P and GM-2P-1 achieved the highest classification accuracies, and the overall classification accuracies (Appendix C) show a trend of reduced accuracy with the addition of more classes (Figure 15). Additionally, the K-means algorithm was more successful at clustering the GM-2P-1 and GM-3P-1 schemes than their counterparts, GM-2P-2 and GM-3P-2 (Table 4). Confusion matrices for each classification scheme (Figure 13) bolster these results and show that the axis classification was the easiest to cluster for each scheme and that off-axis and margin data were commonly misclustered as axis.

Table 4. K-means classification accuracies for each classification scheme.

Classification Scheme	Classification Accuracy
FD-2P	78.98%
FD-3P	48.41%
GM-2P-1	71.34%
GM-2P-2	63.06%
GM-3P-1	55.41%
GM-3P-2	43.31%

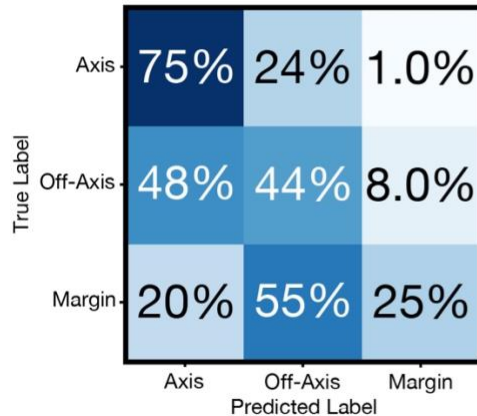
FACIES-DRIVEN

FD-2P



Accuracy: 78.98%

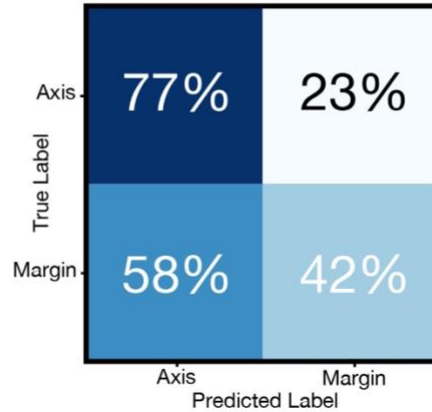
FD-3P



Accuracy: 48.41%

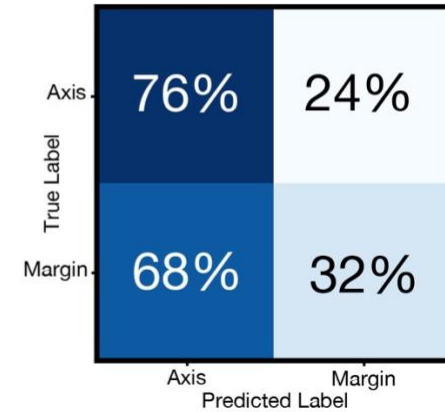
GEOMETRIC

GM-2P-1



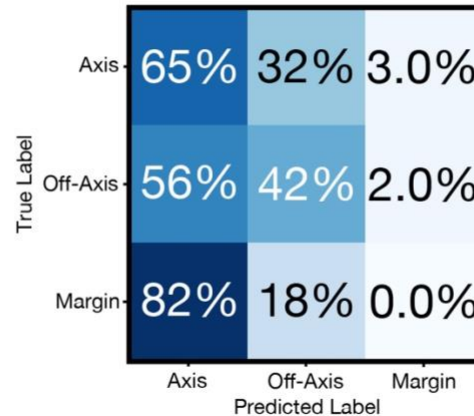
Accuracy: 71.34%

GM-2P-2



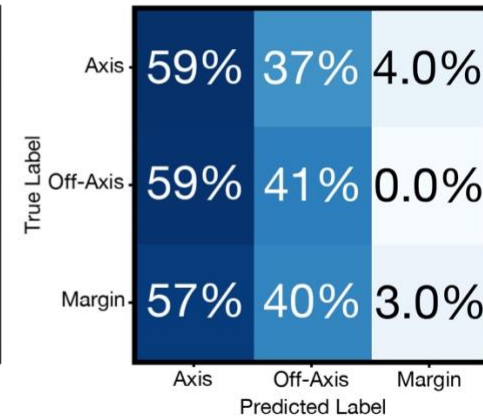
Accuracy: 63.06%

GM-3P-1



Accuracy: 55.41%

GM-3P-2



Accuracy: 43.31%

Figure 13. Normalized confusion matrices for K-means clustering analysis.

4.2.2 Supervised Learning Results

4.2.2.1 Unsupervised vs. Supervised Learning

The full suite of supervised learning algorithms was used to classify the 16-feature dataset. In comparison to the unsupervised learning results, the best-performing supervised learning algorithms outperformed K-means for each classification scheme (Figures 14 and 15; Appendix C). These algorithms and their respective validation accuracies can be viewed in Table 5. The random forest, XGBoost, and ANN algorithms consistently ranked amongst the best-performing algorithms used in these analyses. Random forest and XGBoost are both decision tree-based ensemble learners, which improve their weak learning algorithm by using techniques, such as bagging and boosting. Therefore, they expectedly outperformed their base weak learner and other algorithms, such as KNN, discriminant analysis, and SVMs. Additionally, although it is not an ensemble learner, the ANN is useful for detecting non-linear relationships between data, so it was effective for classifying data in several of the more complicated GM schemes (Table 5; Figure 14).

Table 5. Validation accuracies for best-performing supervised learning algorithms.

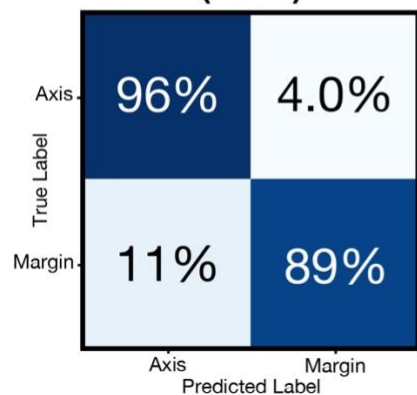
Classification Scheme	Best-Performing Algorithm(s)	Validation Accuracy
FD-2P	ANN	93.63%
FD-3P	Linear SVM & RF	82.80%
GM-2P-1	RF, XGB, & ANN	85.99%
GM-2P-2	RF & ANN	75.16%
GM-3P-1	XGB & ANN	73.89%
GM-3P-2	RF & XGB	62.42%

The improved accuracies for the supervised learning algorithms as compared to K-means (Table 5; Figure 15) highlight the importance of geologic interpretation and guidance. Geologic data is variable and difficult to cluster. Trends are not always clearly defined, and the expertise and

knowledge of a trained geologist can help to decipher complicated channel architecture. This is reflected in the poorer classification accuracies and confusion matrices for the K-means clustering analysis (Table 4; Figure 13). The K-means algorithms performed worse than the supervised learning algorithms for each classification scheme and the addition of more classes reduced the classification accuracy by over 30% (Figure 15).

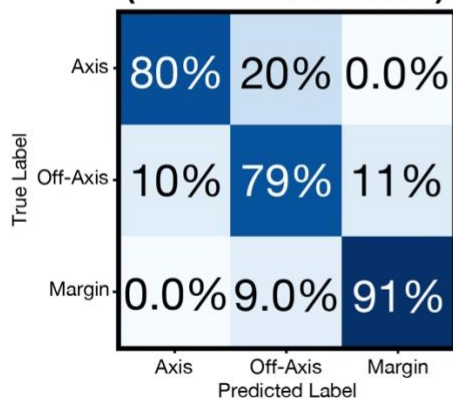
FACIES-DRIVEN

**FD-2P
(ANN)**



Validation Accuracy: 93.63%

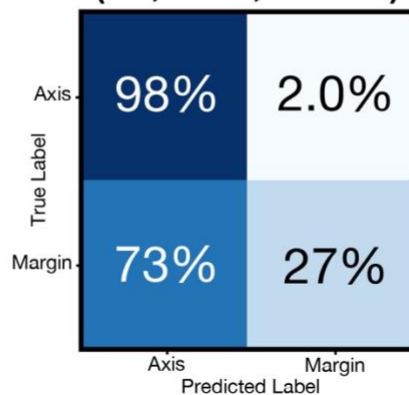
**FD-3P
(Linear SVM & RF)**



Validation Accuracy: 82.80%

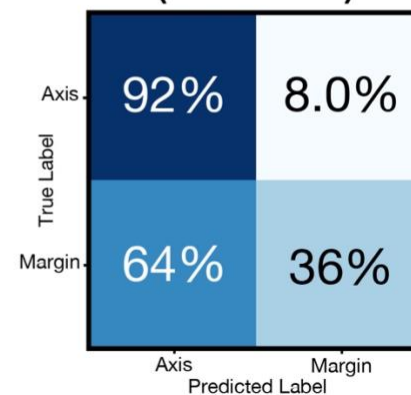
GEOMETRIC

**GM-2P-1
(RF, XGB, & ANN)**



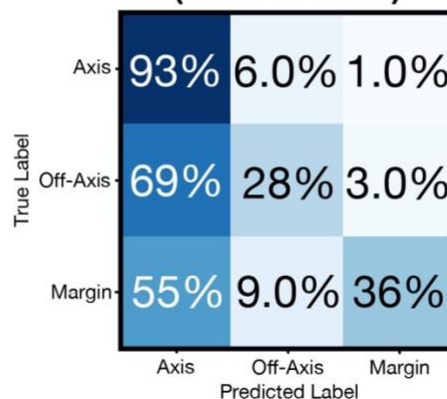
Validation Accuracy: 85.99%

**GM-2P-2
(RF & ANN)**



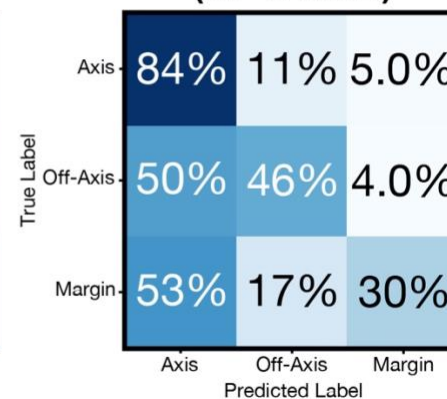
Validation Accuracy: 75.16%

**GM-3P-1
(XGB & ANN)**



Validation Accuracy: 73.89%

**GM-3P-2
(RF & XGB)**



Validation Accuracy: 62.42%

Figure 14. Normalized confusion matrices for best-performing supervised algorithms.

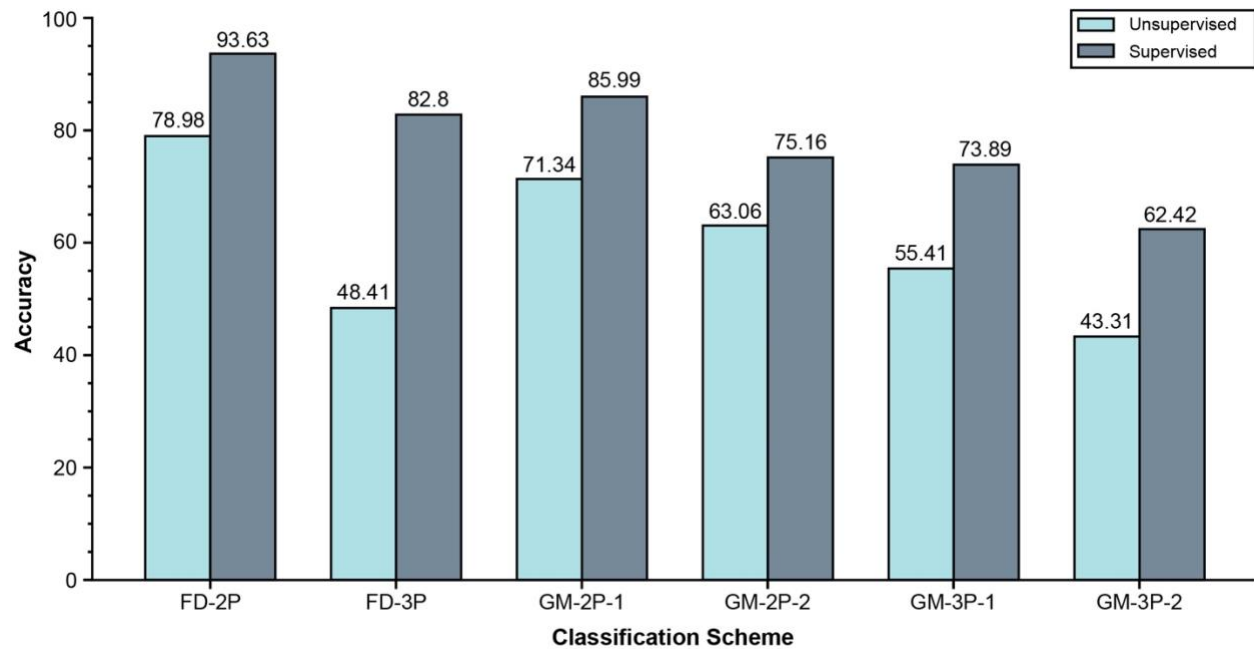


Figure 15. Bar plot of accuracies for the K-means algorithm and the best-performing algorithms from the supervised learning analysis. The supervised learning algorithms outperformed the K-means algorithm for each classification scheme. Accuracy reduces with the addition of more channel positions for both the FD and GM classification schemes.

4.2.2.2 Facies-Driven vs. Geometric Classification Schemes

In regard to classification schemes, the facies-driven schemes outperformed their counterpart geometric schemes in all categories (Figures 15, 16, and 17). This was expected as geologic interpretation and expertise are needed to make sense of outcrop and core data. However, outcrop observations point toward the potential for internal bias within the FD schemes because they account for classes in the data solely from the character of 1D vertical measured section data. It is possible that the FD schemes create an idealized version of each channel position with common characteristics and grouped statistics when in reality diverse styles of architecture potentially exist within channel geobodies.

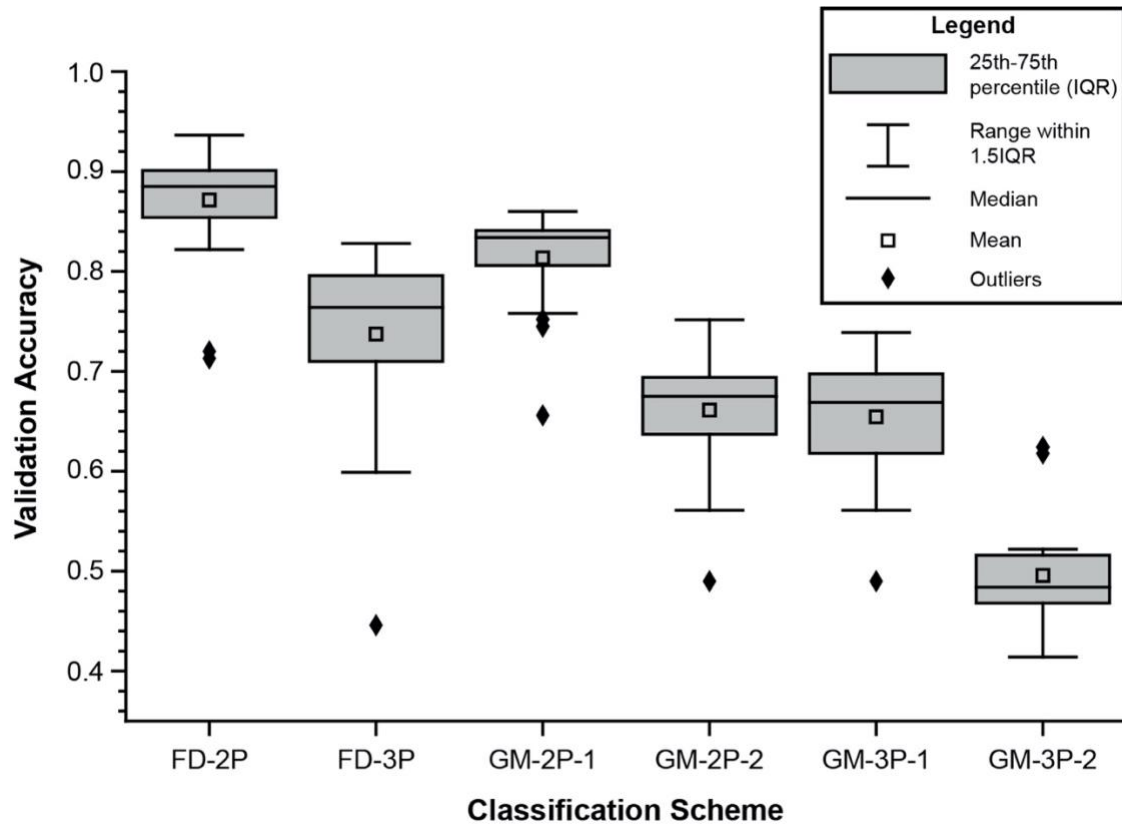


Figure 16. Box and whiskers plot showing distribution of validation accuracies for all supervised learning algorithms.

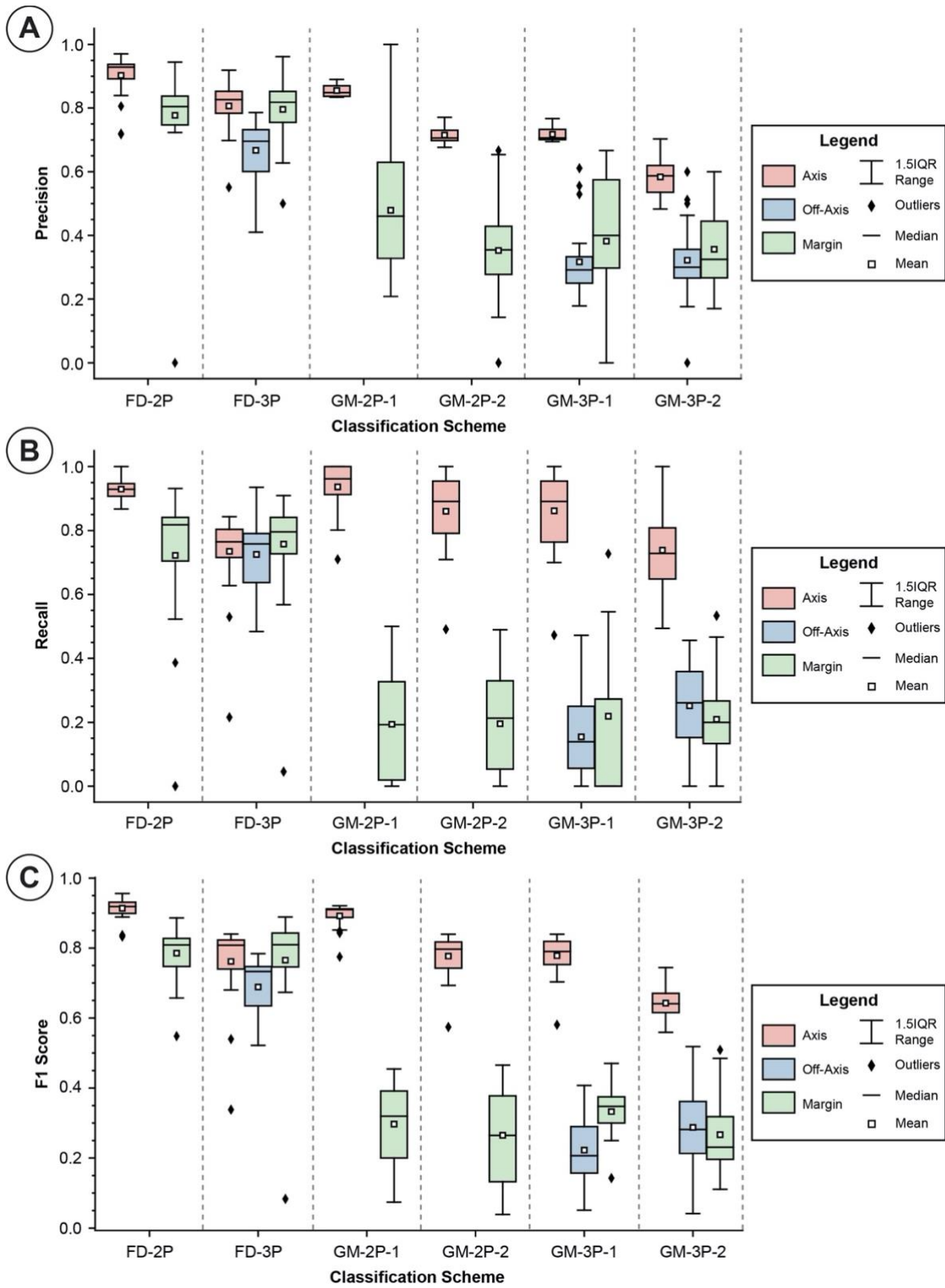


Figure 17. Box and whiskers plots showing distributions of: (A) Precision, (B) Recall, and (C) F1 Score for the supervised learning algorithms.

Southern et al., (2017) proposed the concept of three different architectures that were derived from observations that are counter to the FD classification scheme. These architectures include: (A) Margins with higher proportions of sandstone, higher degrees of amalgamation, lower proportions of siltstone-dominated facies, and higher proportions of turbidites resulting in structureless sandstone (Figure 18A); (B) Margins characterized by thick-bedded sandstones interleaved with less amalgamated sandstone- or siltstone-dominated facies (Figure 18B). Style B siltstone-dominated intervals are generally thicker than those in style A; (C) Margins with thick intervals of siltstone-dominated facies at their base, which are truncated and capped by thick-bedded amalgamated sandstones (Figure 18C).

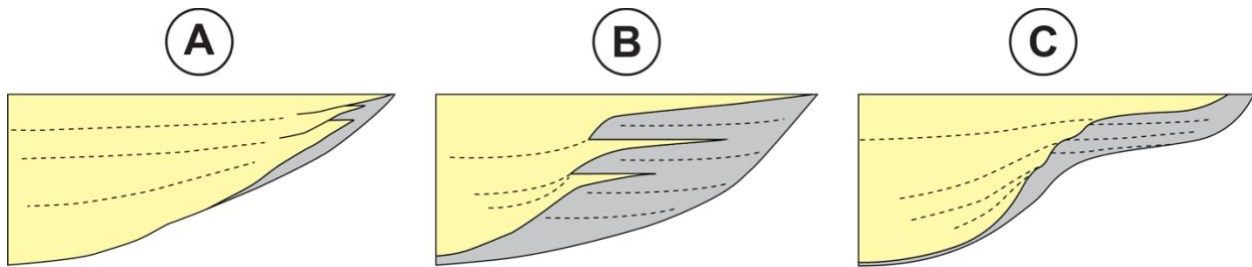


Figure 18. Three different styles of channel margin architecture (Southern et al., 2017). (A) Margins with higher proportions of sandstone, higher degrees of amalgamation, lower proportions of siltstone-dominated facies, and higher proportions of turbidites resulting in structureless sandstone. (B) Margins characterized by thick-bedded sandstones interleaved with less amalgamated sandstone- or siltstone-dominated facies. Style B siltstone-dominated intervals are generally thicker than those in style A. (C) Margins with thick intervals of siltstone-dominated facies at their base, which are truncated and capped by thick-bedded amalgamated sandstones.

As an example, if different margin styles do exist, ranging from more amalgamated, sandier margins to less amalgamated margins with thicker drapes (Southern et al., 2017), a geologist might mistake the former for an axis channel position when looking only at 1D core or measured section data. The geometric schemes on the other hand create an unbiased interpretation that supports the

observations at the outcrop that margin facies are not always the thin-bedded, high proportion of FA4 variety that the facies-driven schemes define. Therefore, the GM schemes would likely capture this diverse heterogeneity across channel positions, which could validate their poorer evaluation metrics and results. This is expanded upon in the hierarchical machine learning workflow in chapter five.

4.2.2.3 Two Positions vs. Three Positions

Overall, the two-position classification schemes (FD-2P and GM-2P-1) achieved the highest validation accuracies, 93.63% and 85.99%, for the FD and GM schemes (Table 5; Figures 14, 15, and 16; Appendix C). These results suggest that it is better to use two-position schemes than three-position schemes in these workflows, as the classifications for the former are more consistent and reproducible. This debate is often referred to as lumping versus splitting, in which a decision has to be made regarding how much detail to consider or take into account. In the context of this problem, a transitional zone (i.e. off-axis) likely exists between axis and margin channel positions, and it is possible to try to capture all of the heterogeneity within a channel geobody by accounting for this zone. However, interpreting off-axis can be difficult and, ultimately, incorporating it as a channel position for models of channel stacking patterns might not have a significant impact on fluid flow and connectivity.

4.2.2.4 Individual Channel Positions

Expanding upon the observations above, in both the unsupervised and supervised learning results, the evaluation metrics were overall consistently higher for the axis classification than either off-axis or margin for each scheme. Axis achieved the highest accuracies for classification (Figures 13 and 14) and the highest precisions, recalls, and F1 scores (Figure 17; Appendix C). These results suggest that axis data contain the most distinct statistics out of all the channel positions.

Considering the input features for the machine learning analysis, axis is well-defined by many of them regardless of which scheme is used: high proportions of FA1, low proportions of FA4, high NTG, little-to-no drupe thickness, lower number of beds, and high amalgamation ratio (Figures 6, 7, and 8). This makes it easy to be consistent when identifying and predicting axis from both outcrops and data.

The statistical distinctness of the axis classification is further validated by the results of the GM classification schemes. In both the two-position and three-position schemes, the schemes with the larger cutoffs for axis, off-axis, and margin—GM-2P-1 and GM-3P-1—achieved higher accuracies than the schemes with reduced cutoffs—GM-2P-2 and GM-3P-2 (Table 5; Figure 16). An explanation of these results is that the larger cutoffs create a wider axis classification and thus generate more axis input data. Axis data are easier to predict than either off-axis or margin data due to their statistical distinctness previously suggested, and therefore, the evaluation metrics are better for those classification schemes.

Conversely to axis, off-axis is a channel position that is difficult to distinguish both in the field and from data. It is the transitional zone in between axis and margin in which facies are shifting laterally from more sandstone-dominated to more siltstone-dominated. This is reflected in the features for off-axis (Figures 6B, 8A, and 8B), and ultimately, these less defined features make it harder for the machine learning algorithms to generate correct classifications. The precision, recall, and F1 score results from the supervised learning analyses (Figure 17; Appendix C) reflect this assertion as the overall distributions of evaluation metrics were lowest for the off-axis classification. Moreover, in both the unsupervised and supervised analyses, off-axis data were consistently misclassified as axis data. In the K-means confusion matrices (Figure 13), 48% of the FD-3P off-axis data, 56% of the GM-3P-1 off-axis data, and 59% of the GM-3P-2 off-axis data

were misclassified as axis. In the best-performing supervised analyses (Figure 14), 10% of the FD-3P off-axis data, 69% of the GM-3P-1 off-axis data, and 50% of the GM-3P-2 off-axis data were misclassified as axis. These results question the statistical significance of off-axis as a channel position. If most of the off-axis data is consistently classified as axis, can the two channel positions be grouped into one classification without loss of resolution?

Even though it is more clearly defined as a channel position than off-axis is, the precision, recall, and F1 score metrics were significantly lower than the axis channel position, particularly for the Geometric schemes (Figure 17; Appendix C). Margins are typically characterized by higher proportions of FA4, lower NTG, lower amalgamation ratio, thicker drapes, and higher number of beds (Figures 6B, 8A, and 8B). However, the machine learning algorithms still struggled to classify this channel position as 20% of the margin data was misclustered as axis and 55% was misclustered as off-axis by the K-means algorithm for the FD-3P scheme (Figure 13). Additionally, 82% and 57% of the margin data were misclustered as axis in the GM-3P-1 and GM-3P-2 schemes, respectively (Figure 13). The confusion matrices are even less favorable in the supervised analyses of the GM schemes (Figure 14) as 55% of the margin data in the GM-3P-1 scheme and 53% of the margin data in the GM-3P-2 scheme were misclassified as axis data. These results reemphasize the idea that margin channel positions are not as homogeneously fine-grained as commonly assumed to be. Instead, as previously mentioned, margins exhibit a high degree of variability in their sedimentological characteristics resulting in different styles of margin architecture (Southern et al., 2017; Figures 6B, 8A, and 8B).

4.3 Refinement of Hypotheses

The observations and results above present new challenges of more heterogeneity in the margin than initially interpreted with cross-over into axis character and question whether the

facies-driven schemes are internally biased by geologic interpretation. This motivates a refined hierarchical machine learning analysis in which Laguna Figueroa axis and off-axis data are clustered separately to test their similarities, and the remaining margin data are clustered into three classes (Figure 19) to test the distinct groupings of margin classes as proposed by Southern et al., (2017). As such, refined hypotheses were created in anticipation of the results of this hierarchical machine learning analysis. The first hypothesis for this workflow conjectures that since the margin data and classification results were variable in the preliminary machine learning analyses, different styles of margin architecture exist and are manifested in the outcrop statistics. The second hypothesis states that because the facies-driven classification schemes create statistical groupings of channel positions through expert geologic interpretation, the geometric classification schemes are better at capturing the true heterogeneity and variant architectural styles within channel margins. These hypotheses are tested using the hierarchical machine learning workflow in chapter five.

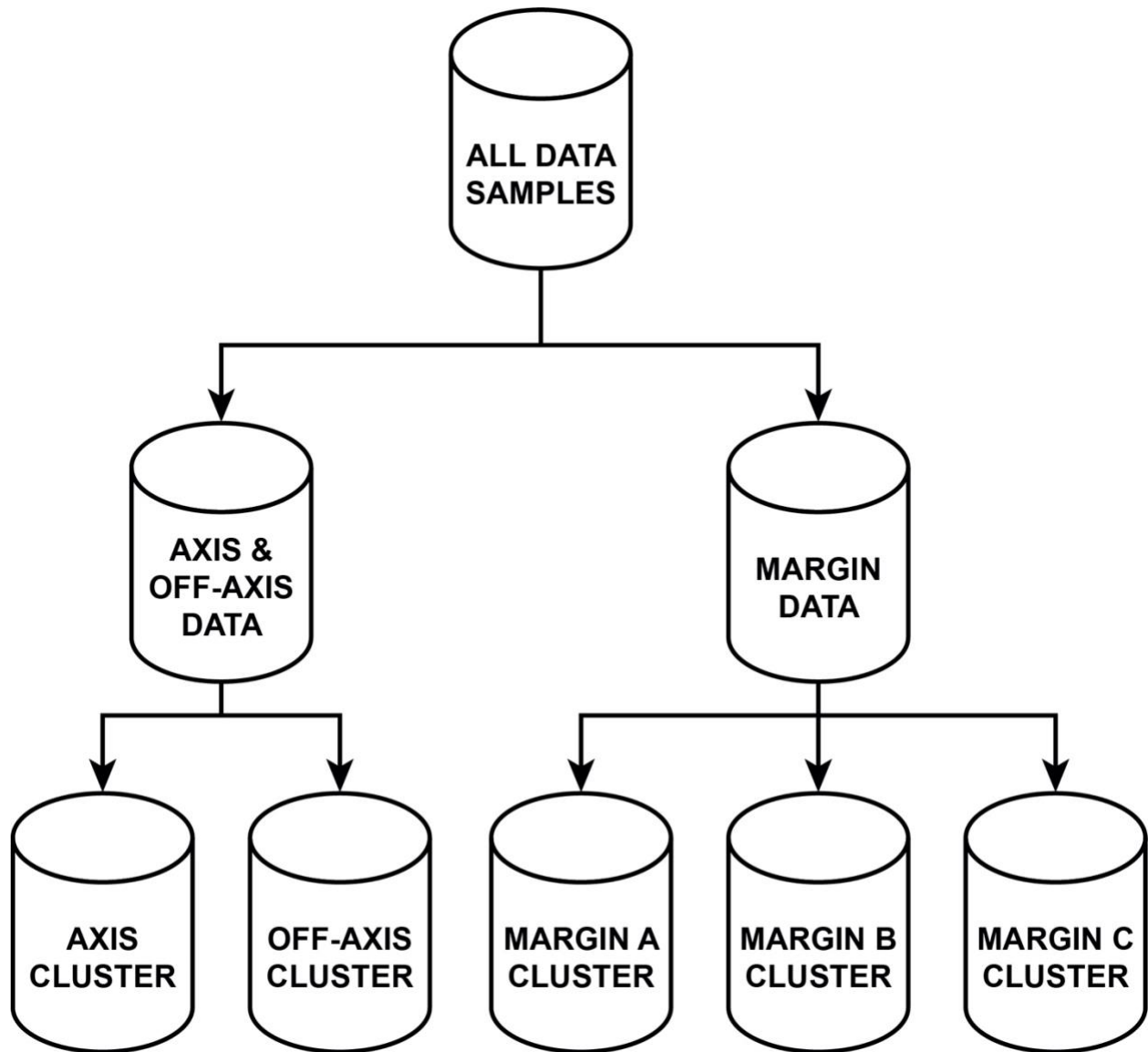


Figure 19. Hierarchical machine learning workflow for partitioning Laguna Figueroa data. All data samples are split into axis/off-axis and margin data in the first split. The axis/off-axis data is subsequently clustered into two classes, and the margin data is subsequently clustered into three clusters—margin A, margin B, and margin C (Southern et al., 2017).

CHAPTER 5: HIERARCHICAL MACHINE LEARNING ANALYSIS

5.1 Methodology

A hierarchical machine learning workflow for partitioning channel outcrop data and analyzing different channel positions and classification schemes is presented in this chapter of this study (Figure 19). Using the three-position classification schemes—FD-3P, GM-3P-1, and GM-3P-2—the 157 data samples from the Laguna Figueroa database were initially split into two partitions. The first partition contains all of the axis and off-axis data combined, and the second partition contains the margin data. K-means was then used to cluster the first partition into two clusters—axis and off-axis—and the second partition into three clusters—margins A, B, and C (Southern et al., 2017). The K-means algorithm was applied for this workflow because there were no available classifications for the three different margin styles for the margin data, thus, supervised learning algorithms could not be used to test for accuracy. Instead, the individual clusters created by the K-means algorithm were analyzed based on their features to draw conclusions about heterogeneity across intra-channel fill, advantages and disadvantages of different classification schemes, and the resulting impacts on fluid flow and reservoir connectivity in stacked deep-water channels.

5.2 Results

The results for the hierarchical machine learning analysis reiterate two main points: 1) axis is the most statistically distinct channel position; and 2) intra-channel fill and architecture are variable, especially in channel margins. Furthermore, the results highlight that the geometric classification schemes capture heterogeneity within channel margins better than the facies-driven classification schemes.

The pairplots—combinations of scatterplots and density plots—of some of the more important features from the Laguna Figueroa database (Figure 12) show that K-means clusters all three schemes similarly for the axis and off-axis channel positions (Figures 20A, 21A, and 22A). The interpreted axis cluster is generally higher in NTG, FA1, and amalgamation ratio, and lower in FA4, drape thickness, and number of beds (Figures 20A, 21A, and 22A). Additionally, axis has the least variability within its features as compared to the other channel positions (Figures 20A, 21A, and 22A). This reaffirms the observations made in chapter four of this study regarding the statistical distinctness of axis as a channel position. This distinctness makes axis the easiest channel position to predict both in the field and from the data. In contrast, off-axis is more variable with greater spreads in its features making prediction more difficult. The main features that best separate the two channel positions are amalgamation ratio and number of beds, as axes are generally more amalgamated with less beds and vice versa (Figures 20A, 21A, and 22A).

Although off-axis is difficult to predict because it is transitional between axis and margin, the greatest variability and thus heterogeneity within a single channel position is seen in the channel margins. The feature distributions for the margin data show that margins A, B, and C are clustered similarly for the FD-3P and GM-3P-2 schemes (Figures 20B and 22B). Firstly, the interpreted margin B clusters for both schemes only contain one data sample. This data sample is characterized as entirely FA4 and drape with a large number of beds. Secondly, the interpreted margin A and margin C clusters for both schemes show that the two margin styles are similar in their characteristics with margin A being slightly sandier and more amalgamated than margin C (Figures 20B and 22B). The primary feature that separates the two styles is the number of beds in each, which is suitable as margin C is characterized by thicker drapes with higher numbers of thin

beds (Southern et al., 2017). Furthermore, this trend of thicker drapes in margin C is better represented by the GM-3P-2 scheme than the FD-3P scheme.

The margin clusters for the GM-3P-1 scheme show the best separation between the three interpreted margin styles (Figure 21B). The interpreted margin A cluster is characterized by higher NTG, amalgamation ratio, and FA1, indicating that it is overall sandier than the other styles. Conversely, the interpreted margin B cluster has higher FA4, thicker drapes, and the highest number of beds, indicating a generally silt-dominated margin. Lastly, the interpreted margin C cluster falls in between the two previous interpreted margin styles as it has a variable range in FA4, NTG, amalgamation ratio, drape, and number of beds.

Based on the results, the GM-3P-1 scheme presents the strongest argument for the existence of three different styles of margin architecture and therefore diverse heterogeneity within the margin channel position. Since this scheme classified data as marginal based on a 16 m cutoff from the base of a channel geobody (Figure 5), it likely accurately captures the true marginal extents of channel geobodies. Contrarily, the GM-3P-2 scheme, which had a cutoff of 11 m for marginal data (Figure 5), likely incorporates too much off-axis and potentially axis data into the margin classification, which results in the statistics for this class resembling more axial positions (Figure 22B).

Overall, these results imply that the facies-driven classification scheme, FD-3P, is biased by what is commonly abstracted as marginal within a channel geobody. The margin data from the FD-3P scheme can be interpreted into three different clusters using the hierarchical machine learning workflow, but the margin A and margin C clusters are very similar (Figure 20B). Therefore, although the facies-driven scheme separates axis, off-axis, and margin data samples well based on features alone (Figure 6B), the hierarchical machine learning results indicate that

the scheme oversimplifies the characteristics of the margin channel position and therefore fails to represent its true heterogeneity. This makes the scheme suitable for use in machine learning analysis, as the geologic interpretation is still valid. However, more emphasis should be placed on the conceptual model for channel margins.

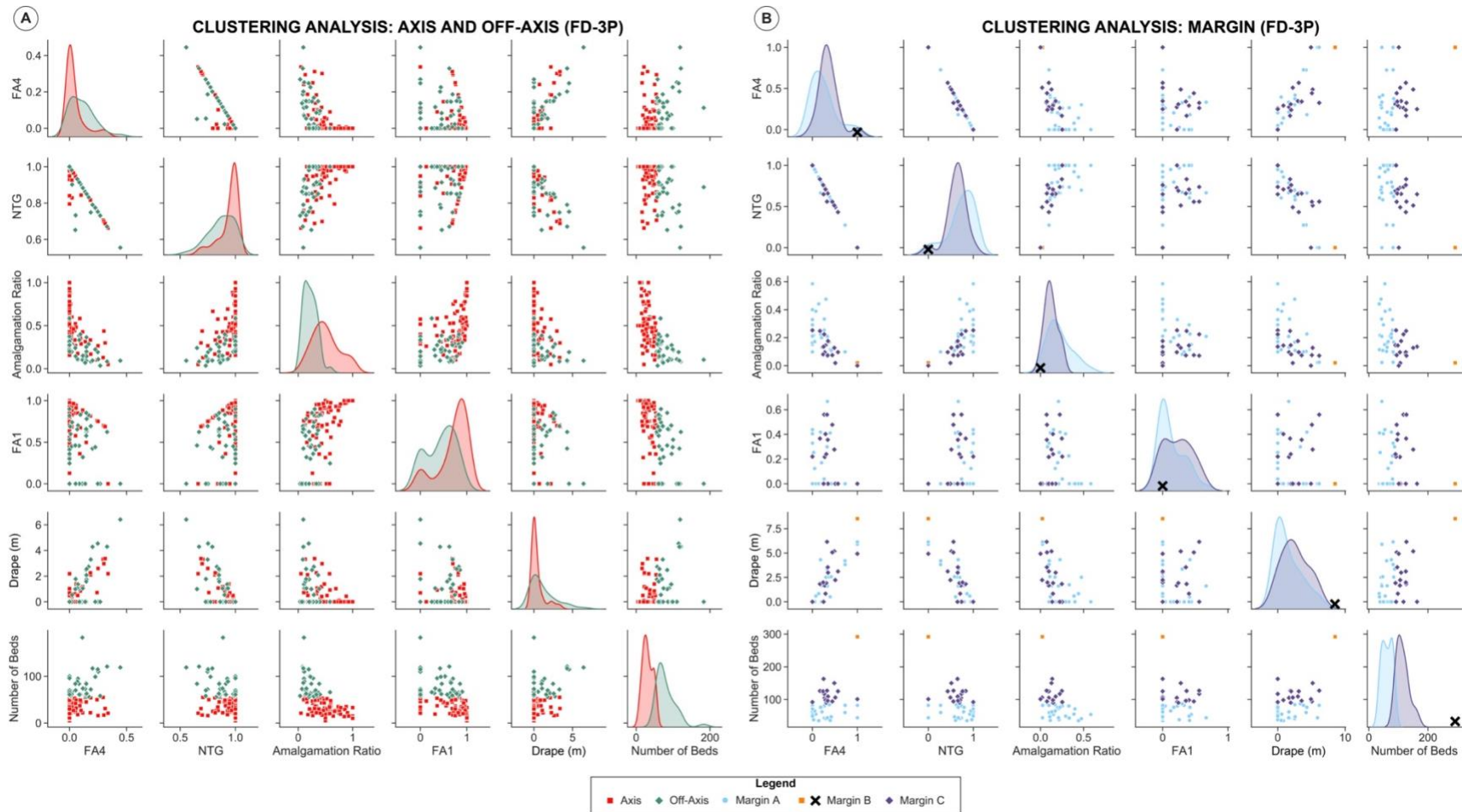


Figure 20. FD-3P feature distributions and scatterplots for axis, off-axis, and margin clusters from hierarchical machine learning workflow. (A) Axis and off-axis clusters. (B) Margin A, margin B, and margin C clusters. Black X symbols represent clusters with zero variance therefore no density plot could be plotted.

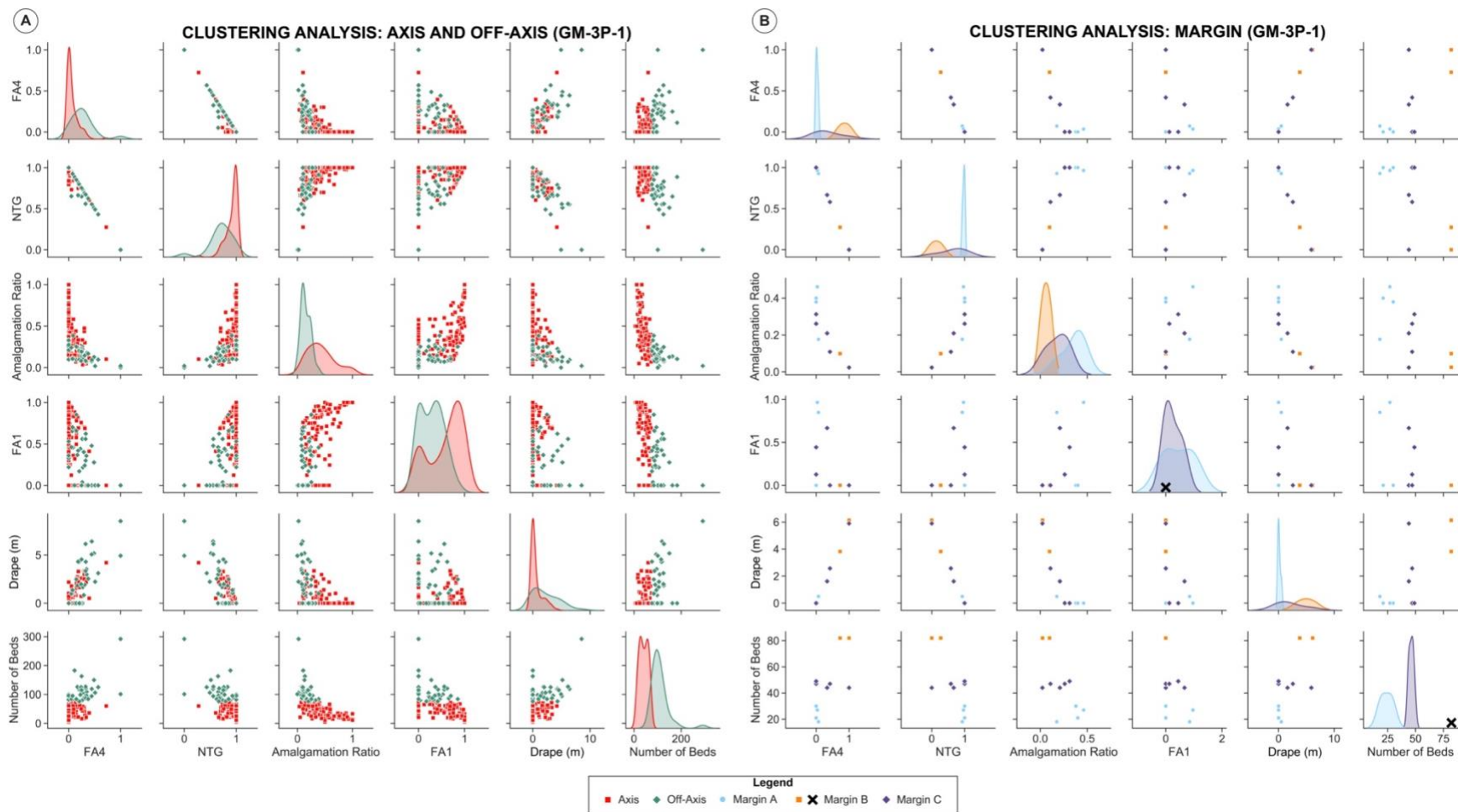


Figure 21. GM-3P-1 feature distributions and scatterplots for axis, off-axis, and margin clusters from hierarchical machine learning workflow. (A) Axis and off-axis clusters. (B) Margin A, margin B, and margin C clusters. Black X symbols represent clusters with zero variance therefore no density plot could be plotted.

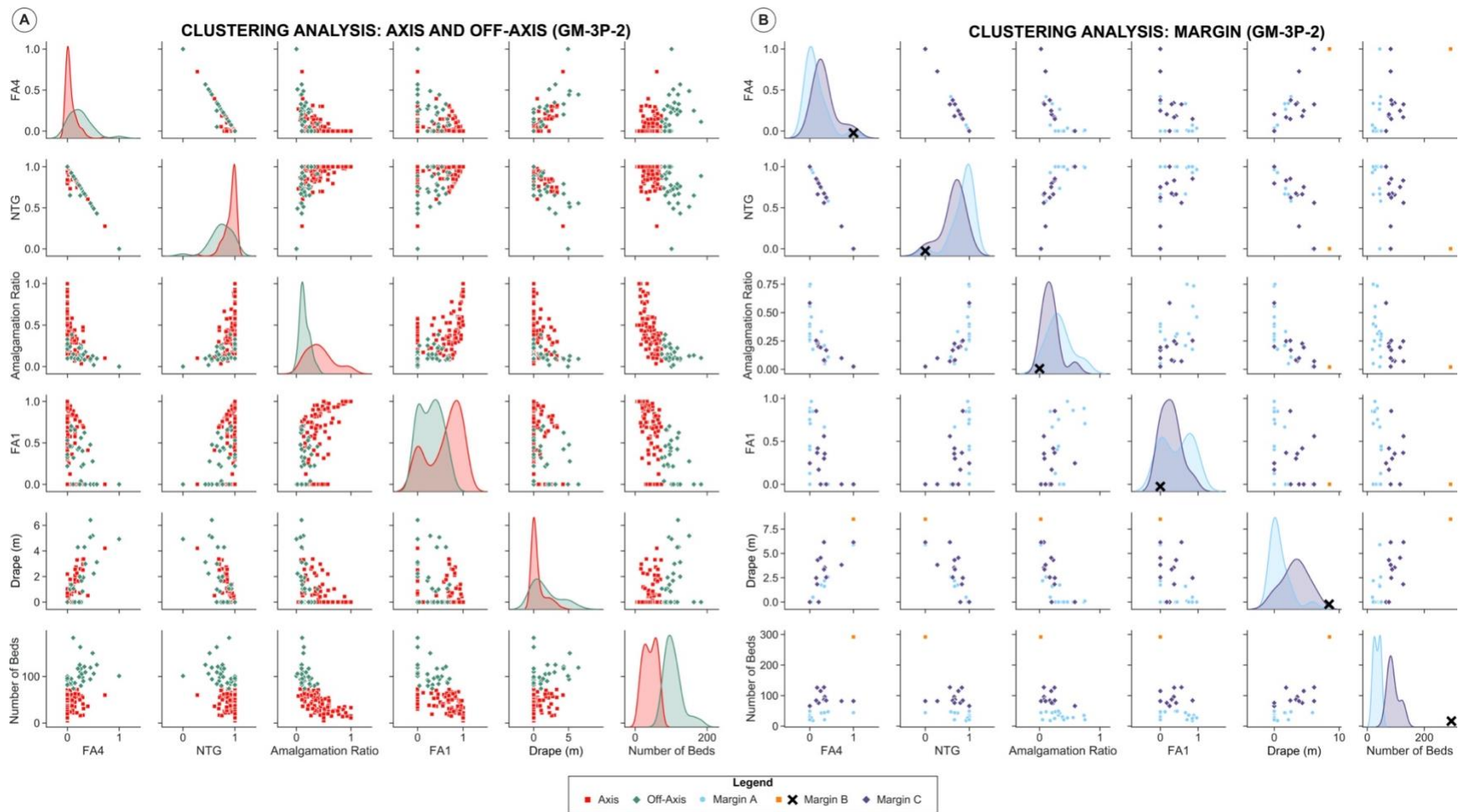


Figure 22. GM-3P-2 feature distributions and scatterplots for axis, off-axis, and margin clusters from hierarchical machine learning workflow. (A) Axis and off-axis clusters. (B) Margin A, margin B, and margin C clusters. Black X symbols represent clusters with zero variance therefore no density plot could be plotted.

CHAPTER 6: DISCUSSION

6.1 Efficacy of Machine Learning Algorithms in Channel Outcrop Analysis

The results of this study validate the use of machine learning algorithms for analyzing and predicting deep-water channel stacking patterns from outcrop data. The best-performing supervised learning algorithms achieved accuracies of over 90% for the FD-2P scheme and over 82% for the FD-3P scheme (Table 5; Appendix C), with the latter producing zero misclassifications of axis data as margin data and vice versa (Figure 14). These results are important when considering the applications of these machine learning algorithms in exploration projects. For example, if a data sample from a high-profile deep-water core is truly a margin classification, but the machine learning algorithms predict it as an axis classification, there will be severe consequences in terms of expected fluid flow and connectivity at that sample location in the wellbore. Furthermore, although there are still errors and uncertainties associated with the supervised learning predictions and classifications can vary between schemes (Figure 23), these slight deficiencies are arguably no worse than the uncertainties that trained geologists encounter when interpreting outcrop or core data of this nature. It is rare, and potentially impossible, to consistently achieve 100% accuracy in interpreting deep-water channel data because a multitude of variables factor into the creation of deep-water strata—tectonics, eustasy, fluid dynamics, sediment supply, time, etc. (Zhang et al., 2017). Additionally, developing these interpretations is often a time consuming and tedious task that can take hours, days, or even weeks. The best-performing machine learning algorithms utilized in this study, on the other hand, achieved accuracies on par with trained geologists in mere seconds, proving their efficacy as a tool in deep-water channel outcrop data analysis.

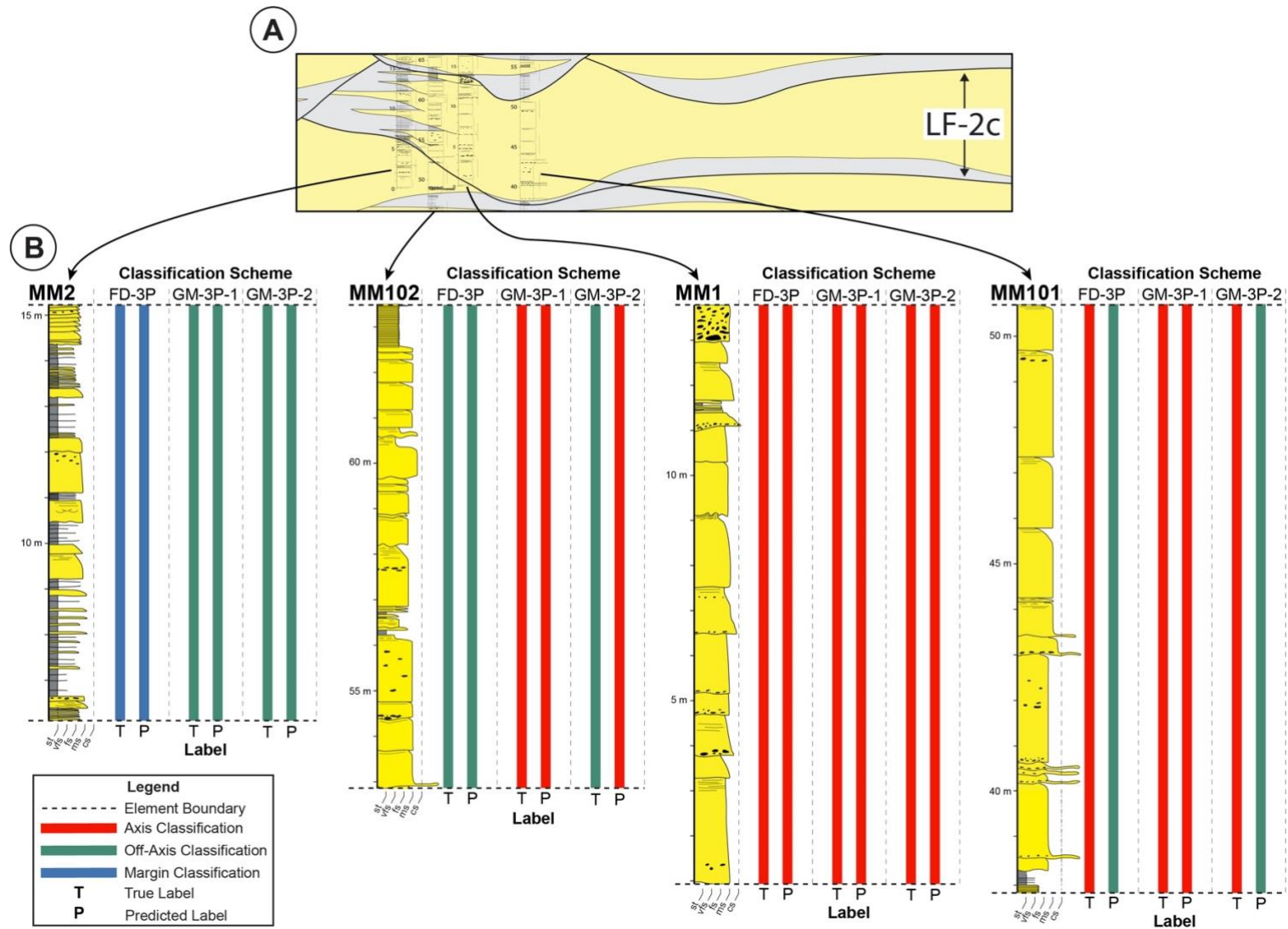


Figure 23. Example of predicted three-position channel position classifications for measured sections from LF-2C (Figure 3A). (A) Clipped image of channel geobody LF-2C from Figure 3A. (B) Measured sections—MM2, MM102, MM1, and MM101—with true and predicted labels for the three-position classification schemes from the best-performing supervised learning algorithms.

However, although these automated analytical techniques have proven to be effective, the importance of the geologic interpretation is still undeniable. Firstly, geologic data is variable and difficult to cluster. Trends are not always clearly defined, and the expertise and knowledge of a trained geologist can help to decipher complicated channel architecture. This is reflected in the classification accuracies and confusion matrices for the K-means clustering analysis (Table 4; Figure 13). The K-means algorithms performed worse than the supervised learning algorithms for each classification scheme and the addition of more classes reduced the classification accuracy by over 30% (Figure 15), thus geologic expertise is still necessary to guide these statistical modeling workflows, especially if data are divided into multiple classes.

Secondly, the FD schemes achieved better evaluation metrics than their GM scheme counterparts for each classification scheme (Figures 15, 16, and 17). The GM schemes were created as objective methods for classifying channel position since the FD schemes were based on geologic expertise and interpretation. However, the GM schemes consistently performed worse than their FD scheme counterparts. Although the GM validation accuracies were greater than 70% for most of the schemes, the equivalent FD schemes still performed better (Table 5).

These results effectively highlight the significance of geologic expertise and knowledge, but there is still uncertainty as to whether the FD schemes are biased by geologic interpretation and what geologists conceptualize as axis, off-axis, and margin. The features for the different channel positions based on the FD schemes show that there is a difference between axis, off-axis, and margin (Figure 6B). However, the hierarchical machine learning workflow applied in this study suggests that this separation between classes is artificially incorporated into the FD scheme by the geologic interpretation. Contrarily, the GM schemes are completely objective and therefore theoretically represent true heterogeneity in each position, but these schemes are esoteric and

potentially only suitable in a study of this nature in which 2D and 3D exposures of channel geobodies are available. It would be impossible to confidently apply the geometric schemes to core data because the vertical and lateral extent of the channel geobodies would be unknown. Therefore, even though the geometric schemes are better at capturing the heterogeneity within intra-channel fill, the best and most feasible method for channel geobody classification is still based on geologic interpretation.

6.2 Variations in Intra-Channel Fill and Impacts on Fluid Flow and Connectivity

Despite the esotericism of the geometric schemes, they were still useful in this study for demonstrating the variability within the margin channel position. Ultimately, the style of margin architecture and channel stacking patterns within a deep-water system can have significant implications for fluid flow and reservoir connectivity (Meirovitz et al., *accepted pending revision*; Jackson et al., 2019). Consider a system characterized by margin A channels with varying degrees of offset ranging from vertically stacked to horizontally stacked (Figure 24). In each stacking scenario, there is still connectivity between the stacked channels due to their sandier axes, more amalgamated beds, and less extensive drapes. However, if the channels in the system are characterized by margin B or margin C architectures, then there are significantly more baffles and barriers to flow than in the previous example (Figure 24). The drape thickness and architecture in margin B channels would cause hydrocarbon recovery and production to be problematic. Even in the vertically stacked scenario, there is minimal contact between the sandy axes of the channels. Margin C channels are more promising for reservoir connectivity than margin B channels, especially in the vertically stacked scenario, but thin drapes could still pose significant issues for flow. Based on these scenarios, special consideration should be given to different styles of margin architecture and intra-channel fill.

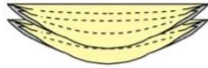
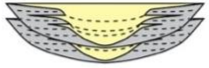
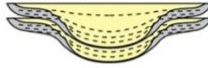

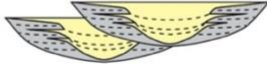
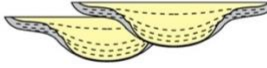



	Margin A	Margin B	Margin C
Vertically Stacked			
Diagonally Stacked			
Horizontally Stacked			

Figure 24. Channel stacking scenarios—vertically stacked, diagonally stacked, and horizontally stacked—for different styles of channel margin architecture.

CHAPTER 7: CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

A database of deep-water channel outcrop statistics from the Tres Pasos Formation at Laguna Figueroa in the Magallanes Basin, Chile, was used to test the efficacy of applying machine learning algorithms to predict subsurface architecture from measured section data (i.e., 1D borehole data). Facies-driven and geometric classification schemes of channel position—axis, off-axis, and margin—were implemented as the desired outputs and prediction labels for the machine learning algorithms. Feature importance, clustering analysis, and supervised learning analyses were performed on the data. The analyses reveal that: 1) features such as minimum and maximum bed thickness, gross, FA2, and FA3 are not as important for defining channel position; (2) axis is the most statistically distinct channel position; (3) the transitional nature of off-axis makes it difficult to classify; and (4) the importance of the geologic interpretation cannot be understated, but the possibility of internal bias should still be considered.

In addition, a hierarchical machine learning workflow was used to expand upon the preliminary analyses and test a hypothesis of different styles of channel margin architecture. These secondary analyses highlight the variability across intra-channel fill and architecture, specifically in channel margins, thus supporting the existence of diverse channel margin architectures. These results have important implications on fluid flow and reservoir connectivity in deep-water channel systems and present more opportunities for the application of data analytics and machine learning in petroleum geoscience and sedimentology.

7.2 Future Work

7.2.1 Modeling Channel Stacking Scenarios

The random forest, XGBoost, and neural network algorithms applied in this study have the ability to produce classification probabilities based on the input features they are provided. These probabilities can be used in a 1D-2D modeling approach to generate different scenarios of channel stacking patterns ranging from least-likely to most-likely and therefore predict connectivity at a wellbore (Figure 25A). Based on the probabilities, if there is a strong anchor at the wellbores delineating the most probable channel stacking patterns, then it's possible to transition more confidently to a 3D inter-wellbore modeling approach (Figure 25B). Ideally, this approach can be used on wells with gamma ray, but the addition of core will help bolster the prediction by providing more statistical data to guide the model.

7.2.2 Classifying Channel Position from Well-Log Data

Measured section data, which is analogous to 1D borehole or core data, provided the foundation for the Laguna Figueroa database used in this study. However, this type of data is not always available in exploration projects due to the cost and time associated with acquiring it. Instead, well-logs, such as gamma ray, density, porosity, and resistivity logs, are typically the standard suites of data available in most exploration projects. On account of this, an enhancement to the applicability of this study could be to train algorithms to detect channel position from solely log responses.

7.2.3 Automatic Detection of Channel Boundaries

The Laguna Figueroa outcrops have been studied extensively over the past couple of decades, which has enhanced the understanding of the deep-water slope channel system and its evolution. This expertise and knowledge allowed researchers to interpret the bases and tops of the

individual channel bodies at Laguna Figueroa, and in turn, the interpretation of these channel boundaries allowed for the partitioning and classification of the different input features used in this study. However, how could a machine learning workflow be applied in an unknown deep-water channel system like those in exploration projects? Can machine learning algorithms be trained to automatically detect channel boundaries from well-log and/or core data? These questions set the stage for future projects with more direct implications on the ability to detect and identify subsurface architecture and heterogeneity.

7.2.4 Data Augmentation

The utility of machine learning algorithms comes from their ability to process large datasets efficiently. Most datasets used in machine learning workflows are deemed Big Data because they have such high volume, variety, velocity, and veracity that they need to be analyzed computationally. However, the Laguna Figueroa database does not meet the requirements to be considered Big Data and some of the classification schemes are imbalanced due to this. This lack of data can impact the machine learning results, making it hard to extract value and conclusions from the analysis. Data augmentation is a technique that is commonly used for data science projects that suffer from issues such as data paucity. It allows users to increase the size and diversity of a dataset without having to retrieve more data (Van Dyk and Meng, 2001). A future project could apply data augmentation methods to the Laguna Figueroa database to increase the size of classes lacking in samples. The machine learning analysis could then be re-run to see whether there are improvements in the evaluation metrics for the machine learning algorithms.

7.2.5 Testing on a Different Deep-Water Channel System

As previously stated in the data augmentation section, the machine learning results in this study could be improved by the addition of more data to strengthen the trends and patterns already

identified or provide more insight into the nature of deep-water channels. Future work could focus on training machine learning algorithms to classify channel position using the Laguna Figueroa database and testing those trained algorithms using data from deep-water channel outcrops from another system. This future work could help improve the evaluation metrics of the algorithms used in this study and validate the use of analogous systems for projects with low data density and/or quality.

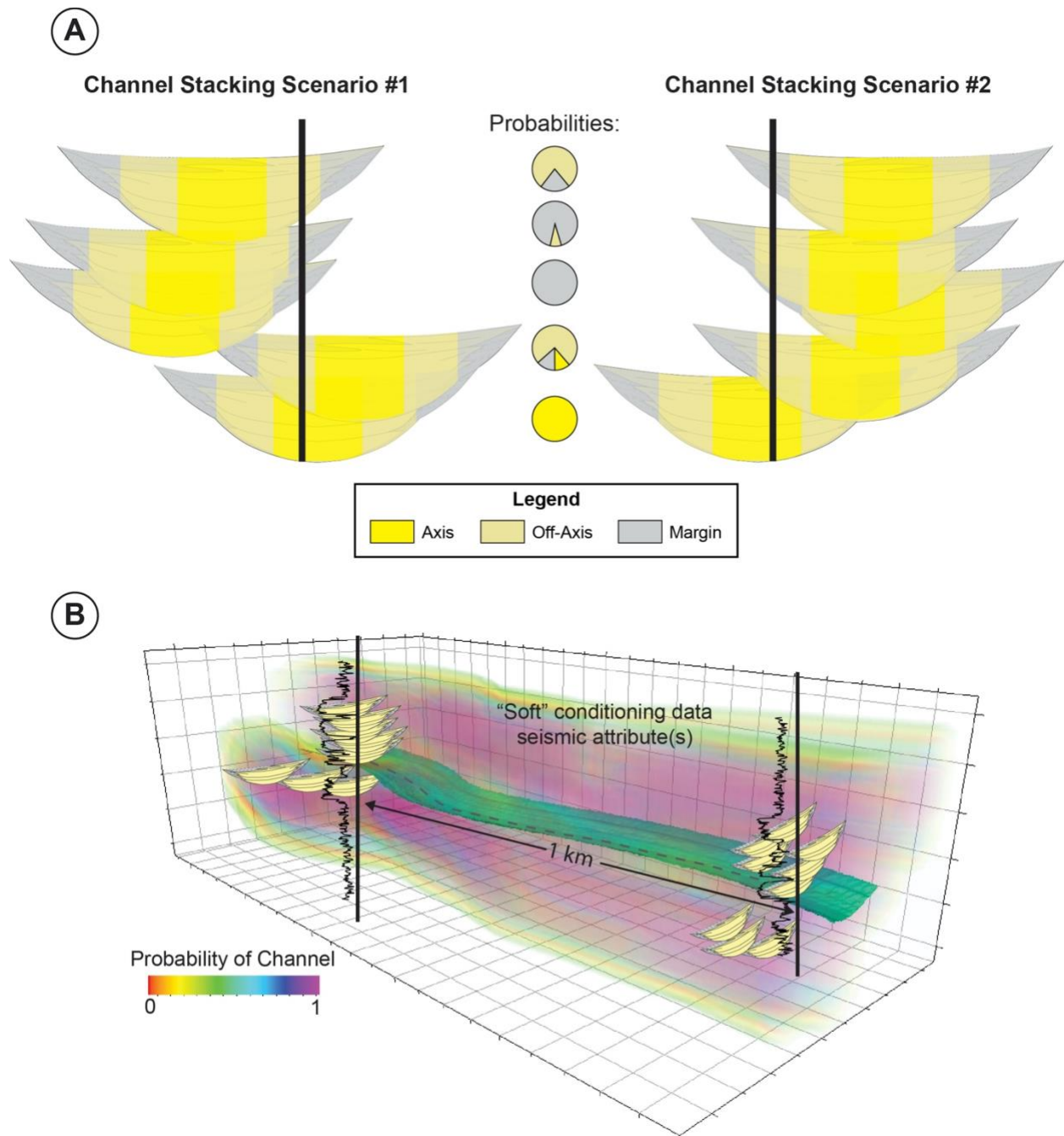


Figure 25. (A) 1D-2D modeling channel stacking scenarios based on most-likely channel positions at a wellbore extracted from machine learning algorithm probabilities. (B) Using strongly anchored or most probable wellbores to predict channel locations between well locations conditioned to seismic attributes (Source: CSS Consortium Material).

REFERENCES

- Abdi, H. and Williams, L.J., 2010, Principal component analysis: *WIREs Computational Statistics*, v. 2, p. 433-459. doi:10.1002/wics.101
- Abuassba, A.O.M., Zhang, D., Luo, X., Shaheryar, A., and Ali, H., 2017, Improving classification performance through an advanced ensemble based heterogeneous extreme learning machines: *Computational Intelligence and Neuroscience*, v. 2017, doi:10.1155/2017/3405463
- Alpak, F.O., Barton, M.D., and Naruk, S.J., 2013, The impact of fine-scale turbidite channel architecture on deep-water reservoir performance: *AAPG Bulletin*, v. 97, p. 251–284, doi:10.1306/04021211067.
- Anderson, C., Forney, E., Hains, D., and Natarajan, A., 2011, Reliable identification of mental tasks using time-embedded EEG and sequential evidence accumulation: *Journal of Neural Engineering*, v. 8, doi:10.1088/1741-2560/8/2/02502.
- Asquith, G.B., Gibson, C.R., Henderson, S.K., Hurley, N.F., and Krygowski, D., 2004, *Basic Well Log Analysis*: American Association of Petroleum Geologists, doi: 10.1306/Mth1682.
- Barton, M., O’Byrne, C., Pirmez, C., Prather, B., van Der Vlugt, F., Alpak, F.O., and Sylvester, Z., 2010, Turbidite channel architecture: recognizing and quantifying the distribution of channel-base drapes using core and dipmeter data: *Dipmeter and Borehole Image Log Technology*, v. 92, p. 195–210, doi:10.1306/13181284M923289.
- Bestagini, P., Lipari, V., and Tubaro, S., 2017, A machine learning approach to facies classification using well-logs: *SEG Technical Program Expanded Abstracts 2017*, p. 2137–2142, doi:10.1190/segam2017-17729805.1.
- Blair, T.C., and McPherson, J.G., 1999, Grain-size and textural classification of coarse sedimentary particles: *Journal of Sedimentary Research*, v. 69, p. 6–19, doi:10.2110/jsr.69.6.
- Brazell, S., Bayeh, A., Ashby, M., and Burton, D., 2019, A machine-learning-based approach to assistive well-log correlation: *Society of Petrophysicists and Well-Log Analysts*, v. 60, p. 469-479.
- Breiman, L., 1994, *Bagging predictors*: Technical Report No. 421: Department of Statistics University of California, Berkeley, California, p. 1-19.
- Bubnova, A., Ors, F., Rivoirard, J., Cojan, I., and Romary, T., 2019, Automatic determination of sedimentary units from well data: *Mathematical Geosciences*, v. 52, p. 213–231, doi:10.1007/s11004-019-09793-w.

- Chen, H., Chiang, R.H.L., and Storey, V.C., 2012, Business intelligence and analytics: From big data to big impact: *MIS Quarterly: Management Information Systems*, v. 36, p. 1165–1188, doi:10.2307/41703503.
- Chen, T., and Guestrin, C., 2016, XGBoost: A scalable tree boosting system: In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, p. 785–794.
- Cheng, B., Xiao, R., Wang, J., Huang, T., and Zhang, L., 2019, High frequency residual learning for multi-scale image classification: *British Machine Vision Association*, p. 1–14.
- Chopra, S., Castagna, J., and Portniaguine, O., 2006, Seismic resolution and thin-bed reflectivity inversion: *CSEG Recorder*, v. 31, p. 19–25, doi:10.1190/1.236994.
- Covault, J.A., Romans, B.W., and Graham, S.A., 2009, Outcrop expression of a continental-margin-scale shelf-edge delta from the Cretaceous Magallanes Basin, Chile: *Journal of Sedimentary Research*, v. 79, p. 523–539, doi:10.2110/jsr.2009.053.
- Dalziel, I.W.D., de Wit, M.J., and Palmer, K.F., 1974, Fossil marginal basin in the southern Andes: *Nature*, v. 250, p. 291–294.
- Daniels, B.G., Hubbard, S.M., Stright, L., and Romans B.W., 2019, Downslope variability in deep-water slope channel fill and stacking patterns: Insights from outcrop and shallow seismic analogs: In *ACE 2019 Annual Convention & Exhibition*.
- Daniels, B.G., Auchter, N.C., Hubbard, S.M., Romans, B.W., Matthews, W.A., and Stright, L., 2018, Timing of deepwater slope evolution constrained by large-n detrital and volcanic ash zircon geochronology, Cretaceous Magallanes Basin, Chile: *Geological Society of America Bulletin*, v. 130, no. 3–4, p. 438–454, doi:10.1130/B31757.1.
- Daniels, B.G., 2019, Multi-scale and geochronologic investigations of Late Cretaceous sediment-routing systems, Magallanes Basin, Chile [Ph.D. thesis]: Calgary, University of Calgary, 319 p.
- Deptuck, M.E., Sylvester, Z., Pirmez, C., and O’Byrne, C., 2007, Migration-aggradation history and 3-D seismic geomorphology of submarine channels in the Pleistocene Benin-major Canyon, western Niger Delta slope: *Marine and Petroleum Geology*, v. 24, p. 406–433, doi:10.1016/j.marpetgeo.2007.01.005.
- Dixon, S.J., and Brereton, R.G., 2009, Comparison of performance of five common classifiers represented as boundary methods: Euclidean Distance to Centroids, Linear Discriminant Analysis, Quadratic Discriminant Analysis, Learning Vector Quantization and Support Vector Machines, as dependent on data structure: *Chemometrics and Intelligent Laboratory Systems*, v. 95, p. 1–17, doi:10.1016/j.chemolab.2008.07.010.

- Fildani, A., and Hessler, A.M., 2005, Stratigraphic record across a retroarc basin inversion: Rocas Verdes-Magallanes Basin, Patagonian Andes, Chile: *Bulletin of the Geological Society of America*, v. 117, p. 1596–1614, doi:10.1130/B25708.
- Fletcher, S.D.T., Macauley, R.V., and Hubbard, S.M., 2011, Characterizing depositional elements of a deep water channel complex using quantitative metrics, Tres Pasos Formation, Southern Chile: 2011 CSPG CWLS Convention, p. 1–4.
- Fletcher, S., 2013, Stratigraphic characterization of a Cretaceous slope channel complex in the Tres Pasos Formation, Arroyo Picana-Laguna Figueroa outcrop belt [M.S. thesis]: Calgary, University of Calgary, 127 p.
- Friedman, J., Hastie, T., and Tibshirani, R., 2000, Additive logistic regression: *The Annals of Statistics*, doi:10.1214/aos/1016218223.
- Friedman, J.H., 2002, Stochastic gradient boosting: *Computational Statistics and Data Analysis*, v. 38, p. 367–378, doi:10.1016/S0167-9473(01)00065-2.
- Freund, Y., and Schapire, R. E., 1996, Experiments with a new boosting algorithm: *Proceedings of the 13th International Conference on Machine Learning*, p. 148–156, doi:10.1.1.133.1040.
- Freund, Y., and Schapire, R.E., 1997, A decision-theoretic generalization of on-line learning and an application to boosting: *Journal of Computer and System Sciences*, v. 55, p. 119–139, doi:10.1006/jcss.1997.1504.
- Fosdick, J.C., Romans, B.W., Fildani, A., Bernhardt, A., Calderón, M., and Graham, S.A., 2011, Kinematic evolution of the Patagonian retroarc fold-and-thrust belt and Magallanes foreland basin, Chile and Argentina, 51°30's: *Bulletin of the Geological Society of America*, v. 123, p. 1679–1698, doi:10.1130/B30242.1.
- Gardner, W.A., 1984, Learning characteristics of stochastic-gradient-descent algorithms: A general study, analysis, and critique: *Signal Processing*, v. 6, p. 113–133, doi:10.1016/0165-1684(84)90013-6.
- Goodfellow, I., Bengio, Y., and Courville, A., 2016, *Deep Learning*: Cambridge, Massachusetts, MIT Press.
- Hall, B., 2016, Facies classification using machine learning: *The Leading Edge*, v. 35, p. 906–909, doi:10.1190/tle35100906.1.
- Han, S., Li, M., and Ren, Q., 2019, Discriminating among tectonic settings of spinel based on multiple machine learning algorithms, *Big Earth Data*, v. 3, p. 67–82, doi:10.1080/20964471.2019.1586074

- Hart, B., 2013. Whither seismic stratigraphy?: Interpretation, v. 1, p. SA3–SA20, doi:10.1190/INT-2013-0049.1.
- Hassani, H., and Silva, E.S, 2018, Big Data: A big opportunity for the petroleum and petrochemical industry: OPEC Energy Review, v. 42, p. 74–89, doi:10.1111/opec.12118.
- Huang, L., Dong, X., and Clee, T.E., 2017, A scalable deep learning platform for identifying geologic features from seismic attributes: Leading Edge, v. 36, p. 249–256, doi:10.1190/tle36030249.1.
- Hubbard, S.M., Fildani, A., Romans, B.W., Covault, J.A., and McHargue, T.R, 2010, High-relief slope clinof orm development: Insights from outcrop, Magallanes Basin, Chile: Journal of Sedimentary Research, v. 80, p. 357–375, doi:10.2110/jsr.2010.042.
- Hubbard, S.M., Covault, J.A., Fildani, A., and Romans, B.W., 2014, Sediment transfer and deposition in slope channels: Deciphering the record of enigmatic deep-sea processes from outcrop: Geological Society of America Bulletin, v. 126, no. 5–6, p. 857–871, doi:10.1130/B30996.1.
- Hubbard, S.M., Romans, B.W., Southern, S.J., Stright, L., Daniels, B.G., Fletcher, S., Jackson, A., Kaempfe, S.A., Macauley, R., Nielson, A., Niquet, D., Meirovitz, C., Pemberton, E., and Reimchen, A.P., 2018, Core- and log-based recognition criteria for deep-water channel bodies: Using outcrops to inform stratigraphic architecture predictions beyond the wellbore: AAPG Annual Convention, Salt Lake City, UT, July 23-25, 2018.
- Jain, A.K., Murty, M.N., and Flynn, P.J, 1999, Data clustering: A review: ACM Computing Surveys, v. 31, p. 264–323, doi:10.1145/331499.331504.
- Jackson, A., Stright, L., Hubbard, S.M., and Romans, B.W, 2019, Static connectivity of stacked deep-water channel elements constrained by high-resolution digital outcrop models: AAPG Bulletin, v. 103, p. 2943–2973, doi:10.1306/03061917346.
- Kassambara, A., 2017. Practical Guide to Principal Component Methods in R: Poland, Statistical Tools for High-Throughput Data Analysis, 29 p.
- Kawabata, D., and Bandibas, J., 2009, Landslide susceptibility mapping using geological data, a DEM from ASTER images and an Artificial Neural Network (ANN): Geomorphology, v. 113, p. 97–109, doi:10.1016/j.geomorph.2009.06.006.
- Kotsiantis, S.B., Zaharakis, I.D., and Pintelas, P.E., 2006, Machine learning: A review of classification and combining techniques: Artificial Intelligence Review, v. 26, p. 159–190, doi:10.1007/s10462-007-9052-3.
- Kotsiantis, S., 2011, Combining bagging, boosting, rotation forest and random subspace methods: Artificial Intelligence Review, v. 35, p. 223–240, doi:10.1007/s10462-010-9192-8.

- LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., and Jackel, L.D., 1989, Backpropagation applied to digit recognition: *Neural Computation*.
- Lecun, Y., Bengio, Y., and Hinton, G., 2015, Deep learning: *Nature*, v. 521, p. 436–444, doi:10.1038/nature14539.
- Lever, J., Krzywinski, M., and Altman, N., 2017, Principal component analysis: *Nature Publishing Group*, v. 14, p. 641–642, doi:10.1038/nmeth.4346.
- Lowe, D.R., Graham, S.A., Malkowski, M.A., and Das, B., 2019, The role of avulsion and splay development in deep-water channel systems: Sedimentology, architecture, and evolution of the deep-water Pliocene Godavari “A” channel complex, India: *Marine and Petroleum Geology*, v. 105, p. 81–99, doi:10.1016/j.marpetgeo.2019.04.010.
- Macauley, R.V., and Hubbard, S.M., 2013, Slope channel sedimentary processes and stratigraphic stacking, Cretaceous Tres Pasos Formation slope system, Chilean Patagonia: *Marine and Petroleum Geology*, v. 41, p. 146–162, doi:10.1016/j.marpetgeo.2012.02.004.
- MacQueen, J., 1967, Some methods for classification and analysis of multivariate observations: *University of California Press, Berkeley, California, Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, v. 1, p. 281–297.
- Martin-Sanchez, F., Verspoor, K., 2014, Big data in medicine is driving big changes: *Yearbook of Medical Informatics*, v. 9, p. 14–20, doi:10.15265/IY-2014-0020.
- McCulloch, W.S., and Pitts, W.A., 1943, Logical calculus of the ideas immanent in nervous Activity: *Bulletin of Mathematical Biophysics*, v. 5, p. 115–133, doi:10.1007/BF02478259.
- McHargue, T.R., and Webb, J.E., 1986, Internal geometry, seismic facies, and petroleum potential of canyons and inner fan channels of the Indus submarine fan: *American Association of Petroleum Geologists Bulletin*, v. 70, p. 161–180, doi:10.1306/94885651-1704-11d7-8645000102c1865d.
- McHargue, T., Pircz, M. J., Sullivan, M. D., Clark, J. D., Fildani, A., Romans, B.W., and Drinkwater, N.J., 2011, Architecture of turbidite channel systems on the continental slope: Patterns and predictions: *Marine and Petroleum Geology*, v. 28, p. 728–743, doi:10.1016/j.marpetgeo.2010.07.008.
- Meirovitz, C., Stright, L., Romans, B.W., and Hubbard, S., 2016, The influence of intra- and inter-channel architecture in selecting optimal gridding for field-scale reservoir simulation: *AAPG Search and Discovery*, #90259, Calgary.
- Martin-Sanchez, F., & Verspoor, K., 2014, Big data in medicine is driving big changes: *Yearbook of Medical Informatics*, v. 9, p. 14–20, doi:10.15265/IY-2014-0020.

- Mohammadpoor, M., and Torabi, F., 2019, Big Data analytics in oil and gas industry: An emerging trend: *Petroleum*, doi:10.1016/j.petlm.2018.11.001.
- Møller, M.F., 1993, A scaled conjugate gradient method for fast supervised learning: *Neural Networks*, v. 6, p. 525-533, doi:10.1016/S0893-6080(05)80056-5.
- Na, B., and Fox, G.C., 2019, Object detection by a super-resolution method and a convolutional neural networks: *Proceedings - 2018 IEEE International Conference on Big Data, Big Data 2018*, p. 2263–2269, doi:10.1109/BigData.2018.8622135.
- Nader, F.H., Browning-Stamp, P., and Lecomte, J.C, 2016, Geological interpretation of 2D seismic reflection profiles onshore Lebanon: Implications for petroleum exploration: *Journal of Petroleum Geology*, v. 39, p. 333–356, doi:10.1111/jpg.12656.
- Oyelade, O.J., Oladipupo, O.O., and Obagbuwa, I. C., 2010, Application of k-Means Clustering algorithm for prediction of students' academic performance: *International Journal of Computer Science and Information Security*, v. 7, p. 292–295.
- Ozcoban, M.S., Isenkul, M.E., Güneş-Durak, S., Ormanci-Acar, T., Övez, S., and Tüfekci, N., 2018, Predicting permeability of compacted clay filtrated with landfill leachate by k-Nearest Neighbors modelling method: *Water Science and Technology*, v. 77, p. 2155–2164, doi:10.2166/wst.2018.139.
- Pemberton, E.A.L., Stright, L., Fletcher, S., and Hubbard, S.M, 2018. The influence of stratigraphic architecture on seismic response: Reflectivity modeling of outcropping deepwater channel units: *Interpretation*, v. 6, p. T783–T808, doi:10.1190/int-2017-0170.1.
- Polikar, R., 2012, Ensemble learning, *in* Zhang, C., and Ma, Y., eds., *Ensemble Machine Learning*: Boston, Massachusetts, Springer, p. 1-34, doi:10.1007/978-1-4419-9326-7.
- Qian, N., 1999, On the momentum term in gradient descent learning algorithms: *Neural Networks*, v. 12, p. 145–151, doi:10.1016/S0893-6080(98)00116-6.
- Romans, B.W., Hubbard, S.M., and Graham, S.A., 2009, Stratigraphic evolution of an outcropping continental slope system, Tres Pasos Formation at Cerro Divisadero, Chile: *Sedimentology*, v. 56, p. 737–764, doi:10.1111/j.1365-3091.2008.00995.x.
- Romans, B.W., Fildani, A., Hubbard, S.M., Covault, J.A., Fosdick, J.C., and Graham, S.A, 2011, Evolution of deep-water stratigraphic architecture, Magallanes Basin, Chile: *Marine and Petroleum Geology*, v. 28, p. 612–628, doi:10.1016/j.marpetgeo.2010.05.002.
- Rosenblatt, F., 1958, The perceptron: A probabilistic model for information storage and organization in the brain: *Psychological Review*, v. 65, p. 386–408, doi:10.1037/h0042519.
- Rosenblatt, F., 1962, *Principles of Neurodynamics*: New York, Spartan Books, 616 p.

- Rumelhart, D.E., Hinton, G.E., and Williams, R.J., 1986, Learning internal representations by error propagation, *in* Rumelhart, D.E., and McClelland, J.L., eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*: Cambridge, Massachusetts, In MIT Press, v.1, p. 318–362.
- Samuel, A. L., 1959, Some studies in machine learning using the game of checkers: *IEEE Transactions on Circuits and Systems for Video Technology*, v. 3, p. 291–301, doi:10.1109/76.257218.
- Saporetto, C.M., da Fonseca, L.G., Pereira, E., and de Oliveira, L.C, 2018, Machine learning approaches for petrographic classification of carbonate-siliciclastic rocks using well-logs and textural information: *Journal of Applied Geophysics*, v. 155, p. 217–225, doi:10.1016/j.jappgeo.2018.06.012.
- Seiffert, C., Khoshgoftaar, T.M., Van Hulse, J., and Napolitano, A., 2010, RUSBoost: A hybrid approach to alleviating class imbalance: *IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans*, v. 40, p. 185–197, doi:10.1109/TSMCA.2009.2029559.
- Shewchuk, J. R., 1994, An introduction to the conjugate gradient method without the agonizing pain: Technical Report CMU-CS-TR-94-125, Carnegie Mellon University.
- Skurichina, M., Duin, R., 2002, Bagging, boosting and the random subspace method for linear classifiers: *Pattern Analysis & Applications*, v. 5, p. 121–135, doi:10.1007/s100440200011.
- Song, Y., Lu, Y., 2015, Decision tree methods: applications for classification and prediction: *Shanghai Archives of Psychiatry*, v. 27, p. 130–135.
- Southern, S.J., Stright, L., Jobe, Z.R., Romans, B., and Hubbard, S., 2017, The stratigraphic expression of slope channel evolution: insights from qualitative and quantitative assessment of channel fills from the Cretaceous Tres Pasos Formation, southern Chile: AAPG Annual Convention, Houston, TX, April 3-5, 2017.
- Tang, M., Xia, L., Wei, D., Yan, S., Du, C., and Cui, H. L., 2017, Distinguishing different cancerous human cells by Raman spectroscopy based on discriminant analysis methods: *Applied Sciences (Switzerland)*, v. 7, doi:10.3390/app7090900.
- Tao, D., Tang, X., and Wu, X., 2006, Asymmetric bagging and random subspace for support vector machines-based relevance feedback in image retrieval: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 28, p. 1088–1099, doi:10.1109/TPAMI.2006.134.
- Tien Bui, D., Pradhan, B., Lofman, O., and Revhaug, I., 2012, Landslide susceptibility assessment in Vietnam using support vector machines, decision tree, and naive bayes models: *Mathematical Problems in Engineering*, v. 2012, doi:10.1155/2012/974638.

- Van Dyk, D., and Meng, X., 2001, The art of data augmentation: *Journal of Computational and Graphical Statistics*, v. 10, p. 1–50, doi:10.1198/10618600152418584.
- Van Wagoner, J.C., Mitchum, R.M., Campion, K.M., and Rahmanian, V.D., 1990, *Siliciclastic Sequence Stratigraphy in Well Logs, Cores, and Outcrops: Concepts for High-Resolution Correlation of Time and Facies*: American Association of Petroleum Geologists, Tulsa, doi:10.1306/Mth7510
- Welling, M., 2005, *Fisher linear discriminant analysis*: Department of Computer Science, University of Toronto.
- Wilson, T. J., 1991, Transition from back-arc to foreland basin development in the southernmost Andes: Stratigraphic record from the Ultima Esperanza District, Chile: *Geological Society of America Bulletin*, v. 103, p. 98–111, doi:10.1130/0016-7606(1991)103<0098:TFBATF>2.3.CO;2
- Wold, S., Esbensen, K.I.M., and Geladi, P., 1987, Principal component analysis: *Chemometrics and Intelligent Laboratory Systems*, v. 2, p. 37-52, doi:10.1016/0169-7439(87)80084-9.2.
- Wrona, T., Pan, I., Gawthorpe, R.L., and Fossen, H., 2018, Seismic facies analysis using machine learning: *Geophysics*, v. 83, p. O83–O95, doi:10.1190/geo2017-0595.1.
- Wyner, A.J., Olson, M., Bleich, J., and Mease, D., 2017, Explaining the success of adaboost and random forests as interpolating classifiers: *Journal of Machine Learning Research*, v. 18, p. 1–33.
- Zhang, H., 2005, The optimality of naïve bayes: *International Journal of Pattern Recognition and Artificial Intelligence*, v. 19, p. 183–198, doi:10.1142/S0218001405003983.
- Zhang, W.B., Duan, T.Z., Liu, Z.Q., Liu, Y.F., Zhao, L., and Xu, R., 2017, Architecture mode, sedimentary evolution and controlling factors of deepwater turbidity channels: A case study of the M Oilfield in West Africa: *Petroleum Science*, v. 14, p. 493–506, doi:10.1007/s12182-017-0181-2.

APPENDIX A: LAGUNA FIGUEROA DATABASE

Table A.1 Drape thickness, net, gross, NTG, and facies proportion statistics for Laguna Figueroa database.

Complex Set	Section Name	Geobody	Drape Thickness (m)	Net (m)	Gross (m)	NTG	FA1	FA2	FA3	FA4
Upper	FIG100	11	6.15	7.82	13.97	0.56	0.56	0.00	0.00	0.32
Upper	FIG100	10	0.00	11.84	11.84	1.00	0.88	0.00	0.12	0.00
Upper	FIG100	8	0.43	21.27	21.85	0.97	0.97	0.00	0.00	0.01
Upper	FIG100	3	0.83	7.10	10.02	0.71	0.40	0.00	0.31	0.29
Upper	FIG100	2	0.00	7.59	8.05	0.94	0.91	0.00	0.03	0.06
Upper	GC1	12	3.54	6.84	10.38	0.66	0.36	0.16	0.15	0.34
Upper	GC10	11	1.18	9.06	10.24	0.88	0.88	0.00	0.00	0.12
Upper	GC2	12	3.36	6.93	10.28	0.67	0.30	0.20	0.18	0.33
Upper	GC3	12	0.20	18.86	19.20	0.98	0.90	0.00	0.08	0.02
Upper	GC4	12	0.00	16.71	16.71	1.00	0.30	0.43	0.12	0.00
Upper	GC6	11	0.11	10.69	10.79	0.99	0.71	0.28	0.00	0.01
Upper	GC7	12	0.54	21.64	24.00	0.90	0.74	0.00	0.16	0.02
Upper	GC8	12	0.00	24.40	24.88	0.98	0.97	0.01	0.00	0.00
Upper	GCNOR	12	2.54	4.21	6.75	0.62	0.00	0.48	0.14	0.38
Upper	GD1	10	0.00	12.50	12.50	1.00	0.55	0.45	0.00	0.00
Upper	GD1	8	0.00	11.59	11.59	1.00	1.00	0.00	0.00	0.00
Upper	GD3	5	0.59	7.03	7.62	0.92	0.00	0.72	0.20	0.08
Upper	GD3	2	0.00	5.22	5.22	1.00	0.00	0.21	0.79	0.00
Upper	GD5GC5	12	1.03	22.54	24.23	0.93	0.93	0.00	0.00	0.00
Upper	GD5GC5	11	0.00	6.31	6.31	1.00	1.00	0.00	0.00	0.00
Upper	GD5GC5	10	0.00	6.32	6.42	0.99	0.71	0.20	0.07	0.01
Upper	GD6KS7	11	0.00	7.32	7.32	1.00	1.00	0.00	0.00	0.00
Upper	GD6KS7	6	0.00	7.24	7.24	1.00	0.44	0.24	0.32	0.00
Upper	GD7	11	0.00	11.79	11.79	1.00	1.00	0.00	0.00	0.00
Upper	GD8	5	0.00	7.62	11.74	0.65	0.00	0.37	0.28	0.25
Upper	KS3	12	8.52	0.00	8.52	0.00	0.00	0.00	0.00	1.00
Upper	MM103	12	0.00	22.76	22.76	1.00	0.90	0.00	0.10	0.00
Upper	MM103	5	0.00	8.68	9.66	0.90	0.00	0.90	0.00	0.10
Upper	P10	8	0.00	15.46	15.46	1.00	0.90	0.00	0.10	0.00

Complex Set	Section Name	Geobody	Drape Thickness (m)	Net (m)	Gross (m)	NTG	FA1	FA2	FA3	FA4
Upper	P10	3	0.00	10.92	11.53	0.95	0.58	0.37	0.00	0.05
Upper	P10	2	0.00	8.90	8.90	1.00	0.49	0.51	0.00	0.00
Upper	P11	5	0.00	6.52	8.72	0.75	0.00	0.00	0.75	0.25
Upper	P11	2	0.00	6.33	6.33	1.00	0.00	0.00	1.00	0.00
Upper	P12	7	1.01	11.06	12.11	0.91	0.46	0.14	0.32	0.08
Upper	P12	6	0.84	5.11	5.95	0.86	0.00	0.86	0.00	0.14
Upper	P13	5	0.00	9.44	9.44	1.00	0.41	0.17	0.42	0.00
Upper	P13	2	0.00	4.34	4.34	1.00	0.00	0.00	1.00	0.00
Upper	P2	3	0.00	11.01	11.01	1.00	0.63	0.37	0.00	0.00
Upper	P3	11	1.08	8.95	10.61	0.84	0.84	0.00	0.00	0.10
Upper	P4	11	0.00	12.26	12.26	1.00	1.00	0.00	0.00	0.00
Upper	P5	11	0.00	10.88	11.48	0.95	0.90	0.00	0.04	0.05
Upper	P5	7	1.67	8.18	11.26	0.73	0.22	0.20	0.31	0.27
Upper	P5	6	0.00	6.25	6.45	0.97	0.25	0.67	0.04	0.03
Upper	P5	5	5.18	5.44	10.62	0.51	0.28	0.00	0.23	0.49
Upper	P6DS5GC9	12	0.00	8.50	8.50	1.00	0.41	0.09	0.51	0.00
Upper	P6DS5GC9	11	0.00	13.05	13.05	1.00	0.74	0.26	0.00	0.00
Upper	P6DS5GC9	7	0.00	11.12	11.12	1.00	0.23	0.18	0.59	0.00
Upper	P6DS5GC9	6	0.51	3.21	5.31	0.60	0.41	0.19	0.00	0.40
Upper	P6DS5GC9	5	0.00	10.17	10.72	0.95	0.12	0.27	0.56	0.05
Upper	P7GD4	11	0.00	12.85	14.48	0.89	0.00	0.03	0.86	0.11
Upper	P7GD4	10	0.00	6.73	6.73	1.00	0.32	0.68	0.00	0.00
Upper	P7GD4	8	0.00	11.00	11.00	1.00	1.00	0.00	0.00	0.00
Upper	P7GD4	5	0.00	8.15	8.15	1.00	0.00	0.56	0.44	0.00
Upper	P8	5	0.00	10.55	10.55	1.00	0.22	0.10	0.68	0.00
Upper	P9	2	0.00	3.64	4.93	0.74	0.04	0.00	0.70	0.26
Upper	VACA3VV3	11	0.42	18.76	20.27	0.93	0.69	0.24	0.00	0.02
Upper	VACA3VV3	10	0.00	9.58	12.05	0.80	0.80	0.00	0.00	0.00
Upper	VACA4VV4	11	4.32	8.54	12.85	0.66	0.42	0.25	0.00	0.18
Upper	VACA4VV4	10	1.83	9.79	11.77	0.83	0.37	0.06	0.41	0.17
Upper	VACA4VV4	8	0.00	17.86	17.86	1.00	1.00	0.00	0.00	0.00

Complex Set	Section Name	Geobody	Drape Thickness (m)	Net (m)	Gross (m)	NTG	FA1	FA2	FA3	FA4
Upper	VACA4VV4	3	0.00	16.29	16.58	0.98	0.93	0.05	0.00	0.02
Upper	VACA4VV4	2	0.00	6.83	6.83	1.00	1.00	0.00	0.00	0.00
Upper	VACA8VV8	11	0.00	8.97	8.97	1.00	0.75	0.25	0.00	0.00
Upper	VACA8VV8	10	0.00	12.69	12.94	0.98	0.45	0.53	0.00	0.02
Upper	VACA8VV8	8	0.00	13.17	13.78	0.96	0.85	0.00	0.11	0.04
Upper	VCM1VV5	11	0.00	16.99	16.99	1.00	0.25	0.58	0.18	0.00
Upper	VCM1VV5	10	0.50	6.42	6.91	0.93	0.85	0.08	0.00	0.07
Upper	VCM1VV5	8	0.00	13.19	13.19	1.00	0.94	0.06	0.00	0.00
Upper	VCM1VV5	3	2.99	8.37	12.82	0.65	0.00	0.00	0.65	0.35
Upper	VCM1VV5	2	0.00	5.99	7.02	0.85	0.50	0.36	0.00	0.15
Upper	VCM2	9	1.92	8.45	12.22	0.69	0.24	0.00	0.45	0.31
Upper	VCM2	8	0.00	11.50	11.50	1.00	0.86	0.06	0.08	0.00
Upper	VCM3	9	0.00	9.41	12.76	0.74	0.56	0.00	0.18	0.26
Upper	VV2	8	0.00	11.14	11.14	1.00	1.00	0.00	0.00	0.00
Upper	VV2	3	0.00	10.69	11.82	0.90	0.83	0.00	0.07	0.10
Upper	VV2	2	0.00	8.18	8.18	1.00	0.75	0.14	0.11	0.00
Upper	VV7	11	0.00	26.42	26.42	1.00	0.53	0.22	0.25	0.00
Upper	VVEDGE	11	5.90	0.00	5.90	0.00	0.00	0.00	0.00	1.00
Upper	VVWB	11	0.00	11.31	14.20	0.80	0.17	0.00	0.63	0.20
Lower	CACH1	8	1.36	17.03	18.39	0.93	0.70	0.07	0.16	0.07
Lower	CACH1	7	5.07	6.38	11.45	0.56	0.48	0.01	0.07	0.44
Lower	CACH1	6	0.00	6.14	6.14	1.00	0.00	0.71	0.29	0.00
Lower	CACH1	3	0.00	17.73	17.73	1.00	0.56	0.15	0.28	0.00
Lower	CACH1	2	0.00	8.13	8.13	1.00	0.85	0.15	0.00	0.00
Lower	CACH2	3	0.00	19.90	20.10	0.99	0.60	0.14	0.25	0.01
Lower	CACH2	2	1.75	4.86	6.61	0.74	0.00	0.74	0.00	0.26
Lower	DS1	9	6.14	0.00	6.14	0.00	0.00	0.00	0.00	1.00
Lower	DS1	8	3.82	1.80	6.58	0.27	0.00	0.00	0.27	0.73
Lower	DS1	7	6.41	8.71	15.70	0.55	0.00	0.24	0.31	0.45
Lower	DS1	6	0.00	13.99	16.88	0.83	0.83	0.00	0.15	0.02
Lower	DS1	2	0.00	7.52	8.22	0.91	0.00	0.77	0.14	0.09

Complex Set	Section Name	Geobody	Drape Thickness (m)	Net (m)	Gross (m)	NTG	FA1	FA2	FA3	FA4
Lower	DYMD1	3	0.00	7.84	7.84	1.00	0.33	0.20	0.47	0.00
Lower	GOLDA	3	1.99	6.33	8.32	0.76	0.00	0.00	0.76	0.24
Lower	GOLDB	3	1.58	9.20	10.78	0.85	0.58	0.28	0.00	0.15
Lower	GOLDC	3	2.11	9.20	11.30	0.81	0.81	0.00	0.00	0.19
Lower	GOLDD	3	1.05	14.43	15.48	0.93	0.65	0.07	0.21	0.07
Lower	KJ1	10	2.57	3.57	6.15	0.58	0.00	0.00	0.58	0.42
Lower	KJ1	9	0.00	9.01	12.33	0.73	0.00	0.53	0.20	0.27
Lower	KJ1	8	1.64	6.08	7.72	0.79	0.00	0.52	0.26	0.21
Lower	KJ1	7	0.00	7.94	7.94	1.00	1.00	0.00	0.00	0.00
Lower	KJ1	5	3.32	7.95	11.27	0.71	0.71	0.00	0.00	0.29
Lower	KJ1	3	3.30	10.38	13.87	0.75	0.69	0.00	0.06	0.19
Lower	KJ1	2	0.00	9.18	9.18	1.00	1.00	0.00	0.00	0.00
Lower	MM1	7	0.00	12.86	12.86	1.00	0.96	0.00	0.04	0.00
Lower	MM101	10	1.62	3.24	4.86	0.67	0.67	0.00	0.00	0.33
Lower	MM101	9	4.30	14.93	22.26	0.67	0.62	0.00	0.05	0.33
Lower	MM101	7	0.62	12.45	13.07	0.95	0.95	0.00	0.00	0.05
Lower	MM101	5	2.97	6.89	9.86	0.70	0.70	0.00	0.00	0.30
Lower	MM101	3	2.23	3.11	6.34	0.49	0.00	0.00	0.49	0.51
Lower	MM102	9	0.00	21.51	21.51	1.00	0.82	0.00	0.18	0.00
Lower	MM102	7	0.00	11.32	11.32	1.00	0.36	0.52	0.12	0.00
Lower	MM102	6	0.72	6.98	9.20	0.76	0.76	0.00	0.00	0.24
Lower	MM102	5	2.60	7.80	10.40	0.75	0.75	0.00	0.00	0.25
Lower	MM102	3	4.21	1.74	6.30	0.28	0.00	0.00	0.28	0.72
Lower	MM102	2	0.00	13.58	15.90	0.85	0.66	0.20	0.00	0.15
Lower	MM2	7	3.12	3.92	9.08	0.43	0.00	0.00	0.43	0.57
Lower	OP1	9	0.00	7.80	7.80	1.00	0.13	0.56	0.31	0.00
Lower	OP2	9	0.00	7.09	7.09	1.00	0.00	0.62	0.38	0.00
Lower	OP2	8	1.75	21.94	25.78	0.85	0.63	0.18	0.04	0.15
Lower	OP2	6	3.37	9.00	13.09	0.69	0.69	0.00	0.00	0.31
Lower	PEQ1	10	0.00	11.41	11.82	0.97	0.97	0.00	0.00	0.03
Lower	PEQ1	8	1.39	22.35	23.74	0.94	0.94	0.00	0.00	0.06

Complex Set	Section Name	Geobody	Drape Thickness (m)	Net (m)	Gross (m)	NTG	FA1	FA2	FA3	FA4
Lower	PEQ1	6	0.00	6.04	7.01	0.86	0.00	0.86	0.00	0.14
Lower	PEQ1	3	0.00	21.99	22.44	0.98	0.88	0.10	0.00	0.02
Lower	PEQ1	2	0.00	14.22	14.22	1.00	0.36	0.43	0.21	0.00
Lower	PEQ2 LOWER	7	4.93	0.00	4.93	0.00	0.00	0.00	0.00	1.00
Lower	PEQ2 LOWER	3	2.19	11.51	13.70	0.84	0.00	0.77	0.07	0.00
Lower	PEQ2 UPPER	10	0.00	9.00	9.00	1.00	0.00	0.88	0.12	0.00
Lower	PEQ2 UPPER	8	0.00	25.02	25.02	1.00	0.98	0.00	0.02	0.00
Lower	PEQ2 UPPER	6	2.51	5.84	8.35	0.70	0.00	0.62	0.08	0.30
Lower	SUBBB1	10	2.48	14.39	16.86	0.85	0.85	0.00	0.00	0.15
Lower	SUBBB1	8	0.00	12.84	14.89	0.86	0.00	0.00	0.86	0.14
Lower	SUBBB1	7	0.00	12.91	12.91	1.00	0.00	0.65	0.35	0.00
Lower	SUBBB1	6	0.92	9.28	10.20	0.91	0.36	0.46	0.09	0.09
Lower	SUBBB1	3	0.00	9.52	9.52	1.00	0.75	0.00	0.25	0.00
Lower	SUBBB1	2	2.08	19.82	23.28	0.85	0.64	0.20	0.00	0.15
Lower	SUBBB2	7	0.99	16.13	24.73	0.65	0.46	0.09	0.10	0.05
Lower	SUBBB2	6	0.00	11.94	11.94	1.00	0.96	0.00	0.04	0.00
Lower	SUBBB3	10	4.55	15.24	20.25	0.75	0.00	0.52	0.23	0.25
Lower	SUBBB3	8	2.19	4.30	6.49	0.66	0.00	0.66	0.00	0.34
Lower	SUBBB3	7	2.26	10.65	13.04	0.82	0.82	0.00	0.00	0.18
Lower	SUBBB3	6	0.00	10.49	10.49	1.00	0.80	0.20	0.00	0.00
Lower	SUBBB3	2	0.49	21.70	22.19	0.98	0.71	0.23	0.04	0.00
Lower	SUBBB4	10	4.29	16.80	21.25	0.79	0.42	0.21	0.16	0.20
Lower	SUBBB4	8	0.70	12.97	13.67	0.95	0.00	0.45	0.50	0.00
Lower	SUBBB4	6	0.00	13.51	13.51	1.00	0.00	0.75	0.25	0.00
Lower	SUBBB4	3	0.85	8.03	8.88	0.90	0.67	0.23	0.00	0.10
Lower	SUBBB4	2	0.00	14.40	18.60	0.77	0.44	0.20	0.14	0.23
Lower	VACA1	10	0.00	19.58	19.69	0.99	0.69	0.00	0.31	0.01
Lower	VACA1	8	2.73	13.57	16.30	0.83	0.83	0.00	0.00	0.17
Lower	VACA1	6	1.78	9.69	13.23	0.73	0.00	0.53	0.21	0.05
Lower	VACA1	3	0.00	9.62	9.62	1.00	0.00	0.82	0.18	0.00
Lower	VACA1	2	2.56	14.51	17.07	0.85	0.78	0.07	0.00	0.15

Complex Set	Section Name	Geobody	Drape Thickness (m)	Net (m)	Gross (m)	NTG	FA1	FA2	FA3	FA4
Lower	VACA2	10	0.00	10.96	10.96	1.00	0.44	0.56	0.00	0.00
Lower	VACA2	8	0.00	25.10	25.12	1.00	0.98	0.00	0.02	0.00
Lower	VACA2	6	1.05	7.94	8.99	0.88	0.32	0.43	0.13	0.12
Lower	VACA2	3	0.00	11.99	11.99	1.00	0.92	0.00	0.08	0.00

Table A.2 Bed statistics and amalgamation ratio for Laguna Figueroa database.

Complex Set	Section Name	Geobody	Min. Bed Thickness (m)	Med. Bed Thickness (m)	Max. Bed Thickness (m)	Number of Beds	Amalgamation Ratio
Upper	FIG100	11	0.01	0.02	2.34	127.00	0.07
Upper	FIG100	10	0.03	0.15	2.06	35.00	0.74
Upper	FIG100	8	0.04	0.57	1.90	35.00	0.88
Upper	FIG100	3	0.01	0.05	1.43	98.00	0.14
Upper	FIG100	2	0.00	0.16	0.86	32.00	0.48
Upper	GC1	12	0.01	0.05	1.50	96.00	0.07
Upper	GC10	11	0.04	0.19	2.27	22.00	0.43
Upper	GC2	12	0.02	0.05	0.98	87.00	0.08
Upper	GC3	12	0.00	0.13	2.93	48.00	0.40
Upper	GC4	12	0.01	0.05	2.52	64.00	0.22
Upper	GC6	11	0.03	0.32	1.59	21.00	0.75
Upper	GC7	12	0.01	0.11	2.55	67.00	0.29
Upper	GC8	12	0.03	0.67	3.61	29.00	0.86
Upper	GCNOR	12	0.01	0.04	1.47	84.00	0.17
Upper	GD1	10	0.01	0.06	2.37	37.00	0.33
Upper	GD1	8	0.11	0.72	2.79	14.00	1.00
Upper	GD3	5	0.01	0.08	0.55	55.00	0.39
Upper	GD3	2	0.01	0.04	0.43	62.00	0.16
Upper	GD5GC5	12	0.01	0.70	3.17	27.00	0.69
Upper	GD5GC5	11	0.05	0.30	1.57	14.00	0.85
Upper	GD5GC5	10	0.03	0.08	1.25	25.00	0.38
Upper	GD6KS7	11	0.02	1.20	3.40	5.00	0.50
Upper	GD6KS7	6	0.02	0.04	1.16	54.00	0.15
Upper	GD7	11	0.02	0.34	3.15	22.00	0.90
Upper	GD8	5	0.01	0.03	1.20	163.00	0.14
Upper	KS3	12	0.00	0.02	0.27	292.00	0.02
Upper	MM103	12	0.01	0.06	3.01	67.00	0.30
Upper	MM103	5	0.01	0.05	1.10	46.00	0.29
Upper	P10	8	0.03	0.29	2.75	25.00	0.58

Complex Set	Section Name	Geobody	Min. Bed Thickness (m)	Med. Bed Thickness (m)	Max. Bed Thickness (m)	Number of Beds	Amalgamation Ratio
Upper	P10	3	0.02	0.05	1.25	42.00	0.27
Upper	P10	2	0.02	0.05	1.15	43.00	0.43
Upper	P11	5	0.01	0.04	0.73	110.00	0.11
Upper	P11	2	0.01	0.04	0.59	72.00	0.21
Upper	P12	7	0.01	0.04	1.18	84.00	0.39
Upper	P12	6	0.02	0.04	1.34	35.00	0.44
Upper	P13	5	0.01	0.04	1.21	70.00	0.32
Upper	P13	2	0.02	0.04	0.53	54.00	0.58
Upper	P2	3	0.02	0.32	3.40	20.00	0.53
Upper	P3	11	0.01	0.08	2.18	27.00	0.35
Upper	P4	11	0.10	0.54	1.63	15.00	0.93
Upper	P5	11	0.03	0.32	2.10	24.00	0.52
Upper	P5	7	0.01	0.03	2.13	105.00	0.10
Upper	P5	6	0.01	0.04	1.30	38.00	0.22
Upper	P5	5	0.01	0.03	1.62	150.00	0.13
Upper	P6DS5GC9	12	0.02	0.08	1.11	49.00	0.29
Upper	P6DS5GC9	11	0.03	0.27	5.28	15.00	0.64
Upper	P6DS5GC9	7	0.02	0.05	1.43	80.00	0.33
Upper	P6DS5GC9	6	0.03	0.06	1.42	36.00	0.11
Upper	P6DS5GC9	5	0.01	0.06	0.79	65.00	0.17
Upper	P7GD4	11	0.01	0.04	0.67	183.00	0.10
Upper	P7GD4	10	0.02	0.12	1.52	19.00	0.50
Upper	P7GD4	8	0.05	0.75	2.04	15.00	1.00
Upper	P7GD4	5	0.02	0.05	1.39	71.00	0.40
Upper	P8	5	0.02	0.04	2.32	92.00	0.25
Upper	P9	2	0.01	0.05	0.27	80.00	0.25
Upper	VACA3VV3	11	0.01	0.47	1.62	45.00	0.43
Upper	VACA3VV3	10	0.03	0.34	1.97	25.00	0.67
Upper	VACA4VV4	11	0.01	0.04	1.99	76.00	0.23
Upper	VACA4VV4	10	0.01	0.03	1.24	126.00	0.25
Upper	VACA4VV4	8	0.07	0.44	2.75	33.00	1.00

Complex Set	Section Name	Geobody	Min. Bed Thickness (m)	Med. Bed Thickness (m)	Max. Bed Thickness (m)	Number of Beds	Amalgamation Ratio
Upper	VACA4VV4	3	0.01	0.25	2.09	43.00	0.48
Upper	VACA4VV4	2	0.12	0.75	1.29	9.00	1.00
Upper	VACA8VV8	11	0.01	0.08	1.17	36.00	0.31
Upper	VACA8VV8	10	0.01	0.05	1.51	62.00	0.20
Upper	VACA8VV8	8	0.00	0.11	1.30	53.00	0.44
Upper	VCM1VV5	11	0.02	0.12	1.97	66.00	0.58
Upper	VCM1VV5	10	0.01	0.05	2.55	18.00	0.18
Upper	VCM1VV5	8	0.00	0.24	2.04	26.00	0.76
Upper	VCM1VV5	3	0.01	0.04	2.35	109.00	0.18
Upper	VCM1VV5	2	0.01	0.04	0.79	45.00	0.16
Upper	VCM2	9	0.01	0.05	1.25	90.00	0.15
Upper	VCM2	8	0.01	0.20	4.11	28.00	0.56
Upper	VCM3	9	0.00	0.05	0.73	117.00	0.22
Upper	VV2	8	0.09	0.76	2.89	11.00	1.00
Upper	VV2	3	0.02	0.07	1.26	61.00	0.40
Upper	VV2	2	0.01	0.15	0.90	33.00	0.78
Upper	VV7	11	0.00	0.08	1.87	100.00	0.21
Upper	VVEDGE	11	0.01	0.04	1.60	44.00	0.02
Upper	VVWB	11	0.01	0.08	2.42	76.00	0.20
Lower	CACH1	8	0.00	0.04	4.17	95.00	0.11
Lower	CACH1	7	0.00	0.04	1.79	94.00	0.08
Lower	CACH1	6	0.00	0.06	1.32	30.00	0.52
Lower	CACH1	3	0.02	0.26	2.25	41.00	0.60
Lower	CACH1	2	0.00	0.02	2.48	25.00	0.25
Lower	CACH2	3	0.01	0.08	3.63	57.00	0.34
Lower	CACH2	2	0.01	0.07	0.77	43.00	0.33
Lower	DS1	9	0.01	0.05	0.37	82.00	0.02
Lower	DS1	8	0.01	0.04	0.61	82.00	0.10
Lower	DS1	7	0.01	0.05	1.37	119.00	0.09
Lower	DS1	6	0.01	0.47	1.70	27.00	0.42
Lower	DS1	2	0.01	0.04	0.75	61.00	0.33

Complex Set	Section Name	Geobody	Min. Bed Thickness (m)	Med. Bed Thickness (m)	Max. Bed Thickness (m)	Number of Beds	Amalgamation Ratio
Lower	DYMD1	3	0.01	0.02	1.06	82.00	0.10
Lower	GOLDA	3	0.00	0.04	0.56	125.00	0.08
Lower	GOLDB	3	0.01	0.04	1.61	61.00	0.13
Lower	GOLDC	3	0.01	0.05	2.30	54.00	0.17
Lower	GOLDD	3	0.01	0.03	2.87	89.00	0.11
Lower	KJ1	10	0.01	0.08	0.57	47.00	0.11
Lower	KJ1	9	0.01	0.04	2.25	58.00	0.04
Lower	KJ1	8	0.01	0.05	1.39	62.00	0.13
Lower	KJ1	7	0.11	0.54	3.89	10.00	1.00
Lower	KJ1	5	0.01	0.08	1.77	55.00	0.13
Lower	KJ1	3	0.01	0.03	4.52	72.00	0.08
Lower	KJ1	2	0.01	0.35	4.71	11.00	0.50
Lower	MM1	7	0.02	0.50	2.35	19.00	0.78
Lower	MM101	10	0.02	0.04	1.46	44.00	0.21
Lower	MM101	9	0.01	0.03	5.89	121.00	0.08
Lower	MM101	7	0.01	0.03	2.38	37.00	0.33
Lower	MM101	5	0.02	0.29	3.39	17.00	0.44
Lower	MM101	3	0.01	0.04	0.42	106.00	0.02
Lower	MM102	9	0.01	0.07	3.58	65.00	0.38
Lower	MM102	7	0.02	0.05	1.48	53.00	0.38
Lower	MM102	6	0.01	0.03	3.41	50.00	0.10
Lower	MM102	5	0.01	0.41	4.72	16.00	0.27
Lower	MM102	3	0.01	0.05	0.58	60.00	0.10
Lower	MM102	2	0.00	0.03	4.62	86.00	0.09
Lower	MM2	7	0.01	0.04	0.87	125.00	0.10
Lower	OP1	9	0.01	0.05	1.05	47.00	0.26
Lower	OP2	9	0.02	0.06	1.22	30.00	0.38
Lower	OP2	8	0.01	0.04	7.71	56.00	0.20
Lower	OP2	6	0.02	0.21	2.68	31.00	0.30
Lower	PEQ1	10	0.01	0.11	3.41	27.00	0.46
Lower	PEQ1	8	0.01	0.09	4.50	49.00	0.48

Complex Set	Section Name	Geobody	Min. Bed Thickness (m)	Med. Bed Thickness (m)	Max. Bed Thickness (m)	Number of Beds	Amalgamation Ratio
Lower	PEQ1	6	0.00	0.04	1.33	55.00	0.31
Lower	PEQ1	3	0.02	0.35	2.52	36.00	0.60
Lower	PEQ1	2	0.00	0.03	2.70	96.00	0.27
Lower	PEQ2 LOWER	7	0.01	0.02	0.24	101.00	0.00
Lower	PEQ2 LOWER	3	0.02	0.06	2.56	34.00	0.58
Lower	PEQ2 UPPER	10	0.02	0.30	2.04	21.00	0.40
Lower	PEQ2 UPPER	8	0.02	0.62	3.63	23.00	0.55
Lower	PEQ2 UPPER	6	0.00	0.06	1.49	41.00	0.48
Lower	SUBBB1	10	0.01	0.03	2.23	84.00	0.25
Lower	SUBBB1	8	0.01	0.06	2.91	73.00	0.07
Lower	SUBBB1	7	0.01	0.04	1.96	52.00	0.16
Lower	SUBBB1	6	0.01	0.18	1.49	37.00	0.58
Lower	SUBBB1	3	0.02	0.04	2.03	30.00	0.21
Lower	SUBBB1	2	0.00	0.03	2.23	64.00	0.37
Lower	SUBBB2	7	0.01	0.04	7.89	84.00	0.24
Lower	SUBBB2	6	0.00	0.08	2.31	30.00	0.38
Lower	SUBBB3	10	0.01	0.05	1.93	115.00	0.19
Lower	SUBBB3	8	0.01	0.13	1.89	21.00	0.05
Lower	SUBBB3	7	0.04	0.27	2.75	22.00	0.33
Lower	SUBBB3	6	0.01	0.17	1.59	25.00	0.67
Lower	SUBBB3	2	0.01	0.09	4.21	52.00	0.53
Lower	SUBBB4	10	0.01	0.04	5.52	117.00	0.09
Lower	SUBBB4	8	0.00	0.04	2.20	47.00	0.15
Lower	SUBBB4	6	0.01	0.05	1.79	65.00	0.38
Lower	SUBBB4	3	0.00	0.03	2.75	49.00	0.35
Lower	SUBBB4	2	0.00	0.04	3.45	99.00	0.22
Lower	VACA1	10	0.01	0.03	3.72	59.00	0.28
Lower	VACA1	8	0.01	0.03	2.22	57.00	0.29
Lower	VACA1	6	0.01	0.04	1.78	70.00	0.14
Lower	VACA1	3	0.00	0.04	1.52	61.00	0.20
Lower	VACA1	2	0.01	0.02	4.82	63.00	0.15

Complex Set	Section Name	Geobody	Min. Bed Thickness (m)	Med. Bed Thickness (m)	Max. Bed Thickness (m)	Number of Beds	Amalgamation Ratio
Lower	VACA2	10	0.01	0.05	1.96	49.00	0.31
Lower	VACA2	8	0.01	0.27	2.80	48.00	0.60
Lower	VACA2	6	0.01	0.03	1.03	74.00	0.15
Lower	VACA2	3	0.01	0.27	2.70	20.00	0.58

Table A.3 Grain size and Phi-scale statistics for Laguna Figueroa database.

Complex Set	Section Name	Geobody	P10 GS (cm)	P50 GS (cm)	P90 GS (cm)	P10 Phi	P50 Phi	P90 Phi
Upper	FIG100	11	0.01	0.35	0.38	1.41	1.53	6.64
Upper	FIG100	10	0.31	0.34	0.42	1.25	1.54	1.68
Upper	FIG100	8	0.33	0.35	0.75	0.42	1.51	1.61
Upper	FIG100	3	0.00	0.33	0.41	1.28	1.60	9.61
Upper	FIG100	2	0.28	0.41	0.62	0.68	1.28	1.81
Upper	GC1	12	0.00	0.46	0.51	0.96	1.13	9.51
Upper	GC10	11	0.26	0.63	1.02	-0.03	0.66	1.94
Upper	GC2	12	0.01	0.24	0.32	1.64	2.04	7.05
Upper	GC3	12	0.29	0.31	0.33	1.58	1.69	1.80
Upper	GC4	12	0.17	0.40	0.54	0.88	1.33	2.57
Upper	GC6	11	0.37	0.46	4.00	-2.00	1.11	1.43
Upper	GC7	12	0.15	0.32	0.34	1.56	1.65	2.76
Upper	GC8	12	0.28	0.32	0.36	1.47	1.66	1.81
Upper	GCNOR	12	0.01	0.25	0.34	1.54	2.01	7.06
Upper	GD1	10	0.20	0.26	0.59	0.76	1.96	2.30
Upper	GD1	8	0.27	0.34	0.66	0.61	1.54	1.88
Upper	GD3	5	0.13	0.21	0.27	1.87	2.23	2.93
Upper	GD3	2	0.01	0.20	0.24	2.08	2.32	7.00
Upper	GD5GC5	12	0.21	0.25	0.31	1.67	1.99	2.27
Upper	GD5GC5	11	0.23	0.28	0.37	1.42	1.84	2.12
Upper	GD5GC5	10	0.18	0.26	0.76	0.39	1.94	2.47
Upper	GD6KS7	11	0.32	0.49	0.86	0.22	1.03	1.65
Upper	GD6KS7	6	0.01	0.24	0.29	1.80	2.05	6.99
Upper	GD7	11	0.25	0.59	1.16	-0.22	0.75	2.00
Upper	GD8	5	0.01	0.21	0.24	2.08	2.23	7.06
Upper	KS3	12	0.01	0.01	0.18	2.51	7.03	7.06
Upper	MM103	12	0.23	0.28	0.29	1.78	1.86	2.12
Upper	MM103	5	0.01	0.25	0.29	1.79	1.99	6.69
Upper	P10	8	0.36	0.46	0.78	0.35	1.11	1.49

Complex Set	Section Name	Geobody	P10 GS (cm)	P50 GS (cm)	P90 GS (cm)	P10 Phi	P50 Phi	P90 Phi
Upper	P10	3	0.17	0.27	0.38	1.40	1.88	2.52
Upper	P10	2	0.17	0.27	0.38	1.39	1.89	2.56
Upper	P11	5	0.01	0.31	0.35	1.53	1.69	6.92
Upper	P11	2	0.01	0.29	0.37	1.45	1.79	6.95
Upper	P12	7	0.11	0.29	0.45	1.17	1.78	3.14
Upper	P12	6	0.01	0.29	0.39	1.35	1.81	6.73
Upper	P13	5	0.01	0.27	0.37	1.42	1.88	6.69
Upper	P13	2	0.01	0.19	0.31	1.69	2.42	6.77
Upper	P2	3	0.24	0.30	0.35	1.51	1.74	2.05
Upper	P3	11	0.01	0.27	0.50	1.00	1.87	6.75
Upper	P4	11	0.20	0.29	0.62	0.69	1.79	2.31
Upper	P5	11	0.28	0.40	0.94	0.09	1.33	1.82
Upper	P5	7	0.13	0.24	0.32	1.67	2.04	2.92
Upper	P5	6	0.01	0.25	0.36	1.49	2.01	6.64
Upper	P5	5	0.01	0.23	0.35	1.50	2.15	7.06
Upper	P6DS5GC9	12	0.01	0.35	0.39	1.34	1.54	6.58
Upper	P6DS5GC9	11	0.37	0.50	1.12	-0.16	1.00	1.44
Upper	P6DS5GC9	7	0.01	0.30	0.46	1.11	1.76	6.42
Upper	P6DS5GC9	6	0.00	0.33	0.47	1.10	1.61	9.62
Upper	P6DS5GC9	5	0.01	0.34	0.41	1.28	1.57	6.50
Upper	P7GD4	11	0.01	0.23	0.27	1.89	2.13	7.05
Upper	P7GD4	10	0.21	0.32	0.43	1.21	1.64	2.23
Upper	P7GD4	8	0.31	0.38	0.80	0.32	1.39	1.70
Upper	P7GD4	5	0.01	0.27	0.30	1.72	1.91	6.70
Upper	P8	5	0.01	0.25	0.31	1.70	1.98	7.06
Upper	P9	2	0.01	0.15	0.25	2.00	2.75	7.06
Upper	VACA3VV3	11	0.21	0.27	0.33	1.62	1.87	2.27
Upper	VACA3VV3	10	0.01	0.35	0.47	1.08	1.52	7.06
Upper	VACA4VV4	11	0.15	0.35	0.45	1.16	1.50	2.78
Upper	VACA4VV4	10	0.01	0.28	0.35	1.52	1.86	6.96
Upper	VACA4VV4	8	0.34	0.41	0.51	0.98	1.28	1.54

Complex Set	Section Name	Geobody	P10 GS (cm)	P50 GS (cm)	P90 GS (cm)	P10 Phi	P50 Phi	P90 Phi
Upper	VACA4VV4	3	0.27	0.34	0.36	1.47	1.54	1.88
Upper	VACA4VV4	2	0.26	0.28	0.35	1.50	1.82	1.93
Upper	VACA8VV8	11	0.25	0.36	0.58	0.79	1.49	1.99
Upper	VACA8VV8	10	0.18	0.24	0.30	1.74	2.06	2.46
Upper	VACA8VV8	8	0.17	0.28	0.53	0.90	1.85	2.52
Upper	VCM1VV5	11	0.20	0.23	0.27	1.89	2.10	2.34
Upper	VCM1VV5	10	0.18	0.23	0.28	1.86	2.10	2.51
Upper	VCM1VV5	8	0.30	0.41	0.55	0.86	1.30	1.74
Upper	VCM1VV5	3	0.01	0.17	0.27	1.91	2.55	7.06
Upper	VCM1VV5	2	0.01	0.28	0.44	1.19	1.85	7.04
Upper	VCM2	9	0.01	0.25	0.35	1.51	1.98	6.98
Upper	VCM2	8	0.27	0.38	0.50	1.01	1.38	1.90
Upper	VCM3	9	0.01	0.35	0.47	1.10	1.50	7.06
Upper	VV2	8	0.26	0.28	0.36	1.49	1.84	1.92
Upper	VV2	3	0.11	0.27	0.34	1.56	1.90	3.21
Upper	VV2	2	0.20	0.27	0.35	1.50	1.88	2.34
Upper	VV7	11	0.19	0.24	0.38	1.39	2.03	2.37
Upper	VVEDGE	11	0.01	0.01	0.52	0.94	7.05	7.06
Upper	VVWB	11	0.01	0.80	1.23	-0.29	0.32	7.05
Lower	CACH1	8	0.18	0.25	0.30	1.71	2.02	2.44
Lower	CACH1	7	0.01	0.24	0.32	1.63	2.04	7.00
Lower	CACH1	6	0.20	0.30	0.32	1.62	1.75	2.34
Lower	CACH1	3	0.22	0.24	0.34	1.55	2.03	2.16
Lower	CACH1	2	0.24	0.30	0.31	1.70	1.73	2.08
Lower	CACH2	3	0.16	0.25	0.28	1.86	2.03	2.64
Lower	CACH2	2	0.01	0.21	0.27	1.92	2.24	7.06
Lower	DS1	9	0.01	0.01	0.26	1.94	7.05	7.06
Lower	DS1	8	0.01	0.01	0.34	1.57	7.05	7.05
Lower	DS1	7	0.01	0.18	0.35	1.53	2.51	7.04
Lower	DS1	6	0.01	0.30	0.35	1.50	1.75	7.04
Lower	DS1	2	0.16	0.28	0.30	1.75	1.85	2.62

Complex Set	Section Name	Geobody	P10 GS (cm)	P50 GS (cm)	P90 GS (cm)	P10 Phi	P50 Phi	P90 Phi
Lower	DYMD1	3	0.01	0.28	0.29	1.80	1.85	7.02
Lower	GOLDA	3	0.01	0.21	0.29	1.77	2.27	7.06
Lower	GOLDB	3	0.16	0.27	0.37	1.44	1.90	2.60
Lower	GOLDC	3	0.16	0.27	0.28	1.82	1.88	2.64
Lower	GOLDD	3	0.17	0.26	0.29	1.80	1.92	2.59
Lower	KJ1	10	0.01	0.10	0.28	1.82	3.26	7.02
Lower	KJ1	9	0.01	0.22	0.28	1.83	2.21	7.03
Lower	KJ1	8	0.01	0.26	0.28	1.82	1.93	7.01
Lower	KJ1	7	0.27	0.28	3.00	-1.58	1.83	1.89
Lower	KJ1	5	0.01	0.28	0.29	1.80	1.85	7.01
Lower	KJ1	3	0.01	0.22	0.28	1.83	2.17	7.01
Lower	KJ1	2	0.23	0.29	0.29	1.78	1.79	2.12
Lower	MM1	7	0.21	0.25	0.29	1.77	2.00	2.23
Lower	MM101	10	0.01	0.22	0.23	2.15	2.18	7.01
Lower	MM101	9	0.01	0.28	0.29	1.81	1.84	7.03
Lower	MM101	7	0.20	0.28	0.29	1.80	1.83	2.32
Lower	MM101	5	0.01	0.28	0.36	1.47	1.82	7.04
Lower	MM101	3	0.01	0.12	0.22	2.19	3.05	7.03
Lower	MM102	9	0.17	0.28	0.29	1.79	1.86	2.57
Lower	MM102	7	0.13	0.27	0.29	1.80	1.87	2.98
Lower	MM102	6	0.01	0.28	0.29	1.79	1.86	7.03
Lower	MM102	5	0.01	0.36	0.36	1.46	1.49	7.01
Lower	MM102	3	0.01	0.01	0.22	2.18	7.02	7.03
Lower	MM102	2	0.01	0.22	0.22	2.17	2.18	6.63
Lower	MM2	7	0.01	0.17	0.25	2.01	2.53	7.06
Lower	OP1	9	0.12	0.21	0.27	1.90	2.26	3.09
Lower	OP2	9	0.15	0.21	0.21	2.22	2.27	2.70
Lower	OP2	8	0.01	0.25	0.28	1.84	2.00	7.06
Lower	OP2	6	0.01	0.21	0.22	2.19	2.26	7.06
Lower	PEQ1	10	0.21	0.22	0.35	1.52	2.20	2.26
Lower	PEQ1	8	0.18	0.21	0.23	2.15	2.22	2.49

Complex Set	Section Name	Geobody	P10 GS (cm)	P50 GS (cm)	P90 GS (cm)	P10 Phi	P50 Phi	P90 Phi
Lower	PEQ1	6	0.01	0.21	0.27	1.88	2.24	6.64
Lower	PEQ1	3	0.21	0.35	0.37	1.43	1.53	2.24
Lower	PEQ1	2	0.18	0.28	0.36	1.49	1.85	2.45
Lower	PEQ2 LOWER	7	0.01	0.01	0.17	2.56	7.05	7.06
Lower	PEQ2 LOWER	3	0.19	0.21	0.27	1.87	2.22	2.39
Lower	PEQ2 UPPER	10	0.16	0.26	0.29	1.77	1.94	2.65
Lower	PEQ2 UPPER	8	0.29	0.36	0.39	1.38	1.48	1.78
Lower	PEQ2 UPPER	6	0.01	0.26	0.37	1.42	1.94	6.99
Lower	SUBBB1	10	0.01	0.21	0.35	1.52	2.23	6.99
Lower	SUBBB1	8	0.01	0.26	0.39	1.35	1.95	7.03
Lower	SUBBB1	7	0.01	0.26	0.36	1.47	1.95	7.06
Lower	SUBBB1	6	0.04	0.37	0.42	1.26	1.44	4.66
Lower	SUBBB1	3	0.17	0.21	0.21	2.22	2.23	2.55
Lower	SUBBB1	2	0.21	0.27	0.34	1.55	1.91	2.25
Lower	SUBBB2	7	0.13	0.18	0.22	2.18	2.46	2.91
Lower	SUBBB2	6	0.16	0.18	0.23	2.12	2.48	2.62
Lower	SUBBB3	10	0.01	0.24	0.30	1.73	2.06	7.04
Lower	SUBBB3	8	0.01	0.25	0.28	1.84	2.01	7.06
Lower	SUBBB3	7	0.01	0.27	0.28	1.83	1.89	7.02
Lower	SUBBB3	6	0.23	0.24	0.25	1.98	2.04	2.13
Lower	SUBBB3	2	0.18	0.27	0.35	1.52	1.88	2.49
Lower	SUBBB4	10	0.01	0.23	0.28	1.84	2.15	7.04
Lower	SUBBB4	8	0.23	0.38	0.43	1.20	1.41	2.13
Lower	SUBBB4	6	0.13	0.35	0.36	1.47	1.53	2.90
Lower	SUBBB4	3	0.12	0.28	0.28	1.83	1.84	3.11
Lower	SUBBB4	2	0.01	0.28	0.29	1.79	1.85	7.05
Lower	VACA1	10	0.21	0.22	0.28	1.83	2.16	2.23
Lower	VACA1	8	0.01	0.25	0.29	1.80	1.97	7.00
Lower	VACA1	6	0.01	0.22	0.28	1.83	2.20	6.65
Lower	VACA1	3	0.01	0.22	0.36	1.46	2.19	7.04
Lower	VACA1	2	0.01	0.22	0.35	1.51	2.20	6.64

Complex Set	Section Name	Geobody	P10 GS (cm)	P50 GS (cm)	P90 GS (cm)	P10 Phi	P50 Phi	P90 Phi
Lower	VACA2	10	0.24	0.29	0.32	1.66	1.79	2.05
Lower	VACA2	8	0.25	0.28	0.32	1.65	1.83	1.98
Lower	VACA2	6	0.01	0.25	0.28	1.84	1.97	7.03
Lower	VACA2	3	0.25	0.29	0.35	1.50	1.78	1.97

Table A.4 Classification schemes and channel position labels for Laguna Figueroa database.

Complex Set	Section Name	Geobody	FD-2P	FD-3P	GM-2P-1	GM-2P-2	GM-3P-1	GM-3P-2
Upper	FIG100	11	3	3	3	3	2	3
Upper	FIG100	10	1	2	3	3	2	3
Upper	FIG100	8	1	1	1	1	1	1
Upper	FIG100	3	3	3	1	1	1	1
Upper	FIG100	2	1	2	1	1	1	1
Upper	GC1	12	3	3	3	3	2	3
Upper	GC10	11	1	1	1	1	1	1
Upper	GC2	12	3	3	3	3	2	3
Upper	GC3	12	1	1	1	1	1	1
Upper	GC4	12	1	2	1	3	2	2
Upper	GC6	11	1	1	1	3	2	3
Upper	GC7	12	1	1	1	1	1	1
Upper	GC8	12	1	1	1	1	1	1
Upper	GCNOR	12	3	3	3	3	2	3
Upper	GD1	10	1	2	1	3	2	2
Upper	GD1	8	1	1	1	3	2	2
Upper	GD3	5	3	3	1	1	1	1
Upper	GD3	2	3	3	1	1	1	1
Upper	GD5GC5	12	1	1	1	1	1	1
Upper	GD5GC5	11	1	1	1	1	1	1
Upper	GD5GC5	10	1	1	1	1	1	2
Upper	GD6KS7	11	1	1	1	1	1	1
Upper	GD6KS7	6	3	3	1	1	1	1
Upper	GD7	11	1	1	1	3	2	2
Upper	GD8	5	3	3	1	1	1	1
Upper	KS3	12	3	3	3	3	2	3
Upper	MM103	12	1	1	1	1	1	1
Upper	MM103	5	1	2	1	1	1	1
Upper	P10	8	1	1	1	1	1	2
Upper	P10	3	1	2	1	1	1	1

Complex Set	Section Name	Geobody	FD-2P	FD-3P	GM-2P-1	GM-2P-2	GM-3P-1	GM-3P-2
Upper	P10	2	1	2	1	1	1	1
Upper	P11	5	1	2	1	1	1	1
Upper	P11	2	3	3	1	1	1	1
Upper	P12	7	1	2	1	1	1	1
Upper	P12	6	3	3	1	1	1	1
Upper	P13	5	1	2	1	1	1	1
Upper	P13	2	3	3	1	1	1	1
Upper	P2	3	1	1	1	1	1	1
Upper	P3	11	1	2	1	1	1	2
Upper	P4	11	1	1	1	1	1	1
Upper	P5	11	1	1	1	1	1	2
Upper	P5	7	3	3	1	1	1	1
Upper	P5	6	3	3	1	1	1	1
Upper	P5	5	3	3	1	1	1	1
Upper	P6DS5GC9	12	3	3	3	3	2	3
Upper	P6DS5GC9	11	1	1	1	1	1	2
Upper	P6DS5GC9	7	3	3	1	1	1	1
Upper	P6DS5GC9	6	3	3	1	1	1	1
Upper	P6DS5GC9	5	3	3	1	1	1	1
Upper	P7GD4	11	1	2	1	1	1	1
Upper	P7GD4	10	1	2	1	1	1	2
Upper	P7GD4	8	1	1	1	1	1	2
Upper	P7GD4	5	3	3	1	1	1	1
Upper	P8	5	3	3	1	1	1	1
Upper	P9	2	3	3	1	1	1	1
Upper	VACA3VV3	11	1	1	1	1	1	1
Upper	VACA3VV3	10	1	1	1	1	1	1
Upper	VACA4VV4	11	3	3	3	3	2	3
Upper	VACA4VV4	10	3	3	1	3	2	3
Upper	VACA4VV4	8	1	1	1	1	1	1
Upper	VACA4VV4	3	1	2	1	1	1	1
Upper	VACA4VV4	2	1	1	1	1	1	1

Complex Set	Section Name	Geobody	FD-2P	FD-3P	GM-2P-1	GM-2P-2	GM-3P-1	GM-3P-2
Upper	VACA8VV8	11	1	2	1	3	2	3
Upper	VACA8VV8	10	1	2	1	1	1	2
Upper	VACA8VV8	8	1	1	1	1	1	1
Upper	VCM1VV5	11	1	2	1	3	2	3
Upper	VCM1VV5	10	1	2	3	3	3	3
Upper	VCM1VV5	8	1	1	1	3	2	2
Upper	VCM1VV5	3	3	3	1	1	1	1
Upper	VCM1VV5	2	1	2	1	1	1	1
Upper	VCM2	9	3	3	1	1	1	1
Upper	VCM2	8	1	1	3	3	2	3
Upper	VCM3	9	3	3	1	1	1	1
Upper	VV2	8	1	1	1	3	2	2
Upper	VV2	3	1	2	1	1	1	1
Upper	VV2	2	1	2	1	1	1	1
Upper	VV7	11	1	1	1	1	1	1
Upper	VVEDGE	11	3	3	3	3	3	3
Upper	VVWB	11	3	3	3	3	2	3
Lower	CACH1	8	1	2	1	3	2	2
Lower	CACH1	7	3	3	1	1	1	1
Lower	CACH1	6	1	2	1	1	1	2
Lower	CACH1	3	1	1	1	1	1	1
Lower	CACH1	2	1	2	3	3	2	3
Lower	CACH2	3	1	1	1	1	1	1
Lower	CACH2	2	3	3	3	3	2	3
Lower	DS1	9	3	3	3	3	3	3
Lower	DS1	8	3	3	3	3	3	3
Lower	DS1	7	1	2	1	1	1	2
Lower	DS1	6	1	1	1	1	1	1
Lower	DS1	2	1	2	1	3	2	2
Lower	DYMD1	3	3	3	1	1	1	2
Lower	GOLDA	3	3	3	1	3	2	2
Lower	GOLDB	3	1	2	1	1	1	1

Complex Set	Section Name	Geobody	FD-2P	FD-3P	GM-2P-1	GM-2P-2	GM-3P-1	GM-3P-2
Lower	GOLDC	3	1	2	1	1	1	1
Lower	GOLDD	3	1	2	1	1	1	1
Lower	KJ1	10	3	3	3	3	3	3
Lower	KJ1	9	1	2	1	3	2	2
Lower	KJ1	8	3	3	1	3	2	2
Lower	KJ1	7	1	1	1	1	1	1
Lower	KJ1	5	1	2	1	1	1	2
Lower	KJ1	3	1	2	1	1	1	1
Lower	KJ1	2	1	1	1	1	1	2
Lower	MM1	7	1	1	1	1	1	1
Lower	MM101	10	3	3	3	3	3	3
Lower	MM101	9	1	1	1	1	1	1
Lower	MM101	7	1	1	1	1	1	1
Lower	MM101	5	1	2	1	1	1	1
Lower	MM101	3	3	3	1	1	1	2
Lower	MM102	9	1	1	1	1	1	1
Lower	MM102	7	1	2	1	1	1	2
Lower	MM102	6	1	2	1	3	2	2
Lower	MM102	5	1	2	1	1	1	1
Lower	MM102	3	3	3	1	1	1	2
Lower	MM102	2	1	2	1	1	1	2
Lower	MM2	7	3	3	1	3	2	2
Lower	OP1	9	1	2	3	3	3	3
Lower	OP2	9	1	2	3	3	3	3
Lower	OP2	8	1	1	1	1	1	1
Lower	OP2	6	1	2	1	1	1	2
Lower	PEQ1	10	1	2	3	3	3	3
Lower	PEQ1	8	1	1	1	1	1	1
Lower	PEQ1	6	1	2	1	1	1	2
Lower	PEQ1	3	1	1	1	1	1	1
Lower	PEQ1	2	1	2	1	1	1	2
Lower	PEQ2 LOWER	7	3	3	1	1	1	1

Complex Set	Section Name	Geobody	FD-2P	FD-3P	GM-2P-1	GM-2P-2	GM-3P-1	GM-3P-2
Lower	PEQ2 LOWER	3	1	2	1	1	1	2
Lower	PEQ2 UPPER	10	1	2	3	3	3	3
Lower	PEQ2 UPPER	8	1	1	1	1	1	1
Lower	PEQ2 UPPER	6	3	3	1	1	1	2
Lower	SUBBB1	10	1	2	3	3	2	3
Lower	SUBBB1	8	1	2	1	3	2	2
Lower	SUBBB1	7	1	2	1	1	1	1
Lower	SUBBB1	6	1	1	1	1	1	1
Lower	SUBBB1	3	1	2	1	1	1	2
Lower	SUBBB1	2	1	1	1	1	1	1
Lower	SUBBB2	7	1	2	1	1	1	1
Lower	SUBBB2	6	1	1	1	1	1	1
Lower	SUBBB3	10	1	2	3	3	2	3
Lower	SUBBB3	8	1	2	3	3	2	3
Lower	SUBBB3	7	1	2	1	1	1	1
Lower	SUBBB3	6	1	1	1	1	1	2
Lower	SUBBB3	2	1	1	1	1	1	1
Lower	SUBBB4	10	1	2	1	1	1	2
Lower	SUBBB4	8	1	1	1	1	1	1
Lower	SUBBB4	6	1	1	1	1	1	1
Lower	SUBBB4	3	1	2	1	3	2	2
Lower	SUBBB4	2	1	2	1	1	1	2
Lower	VACA1	10	1	1	1	3	2	2
Lower	VACA1	8	1	1	1	1	1	1
Lower	VACA1	6	1	2	1	1	1	1
Lower	VACA1	3	1	2	1	3	2	2
Lower	VACA1	2	1	2	1	1	1	2
Lower	VACA2	10	1	2	3	3	3	3
Lower	VACA2	8	1	1	1	1	1	1
Lower	VACA2	6	1	2	1	1	1	2
Lower	VACA2	3	1	2	1	1	1	2

Classification Schemes: 1 = Axis; 2 = Off-Axis; 3 = Margin

APPENDIX B: PYTHON CODES

Kmeans.ipynb

```
#Filename: Kmeans.ipynb
#Author: Noah Vento
#Description: Cluster data using K-means clustering algorithm

#Import libraries
%matplotlib inline
import pandas as pd
import numpy as np
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import IPython.display as ipd # for display and clear_output
import time # for sleep

#Import Excel spreadsheet of data
Laguna_Fig = pd.read_excel('/Users/noahvento/Desktop/CSS_Noah/Database/M
L_Fig_Stats_Table.xlsx')

#Partition data into desired features
data = Laguna_Fig[['NTG', 'F4_Prop', 'F3_Prop', 'F2_Prop', 'F1_Prop',
                  'Gross', 'Drape', 'AR', 'Net', 'Num_Beds', 'P10_Phi_1cm',
                  'P50_Phi_1cm', 'P90_Phi_1cm', 'BT_min',
                  'BT_med', 'BT_max']]

#Make X have 16 columns and as many rows as needed to hold the values of
data
#Samples are in rows, and the features of each sample are in the columns
#The ith row of X is Sample i whose correct target output is in row i of
T
X = np.array(data).reshape((-1,16))

#Specify desired classification scheme and outputs
#(e.g., three-position classification scheme, where Axis = 1; Off-Axis =
2; Margin = 3)
Pos_3 = Laguna_Fig[['Pos_3']]

#Make T have one column and as many rows as needed to hold the values of
Pos_3
T = np.array(Pos_3).reshape((-1,1))

#Convert T outputs to 0, 1, and 2 instead of 1, 2, and 3 (Python)
def pos_conversion(self):
    T[T == 1] = 0 # convert Axis from 1 to 0
    T[T == 2] = 1 # convert Off-Axis from 2 to 1
    T[T == 3] = 2 # convert Margin from 3 to 2

pos_conversion(T)

#Import KMeans function from Scikit-learn
from sklearn.cluster import KMeans

#Perform KMeans with 3 clusters (clusters match the number of unique cla
sses in T)
kmeans = KMeans(n_clusters=3) # performing kmeans with 3 clusters
kmeans.fit(X)
```

```

y_kmeans = kmeans.predict(X) #Use Kmeans to predict on X data

#Reshape y_kmeans to be able to compare with T
k_results = np.array(y_kmeans).reshape((-1,1))

#Calculate accuracy by comparing predicted cluster labels with true labels from T
#This will require some interpretation of the generated clusters based on features
#since kmeans is an unsupervised algorithm and does not know about the data's true labels
kmeans_acc = (np.sum(T == k_results)/np.size(T))*100

```

Random_Forest.ipynb

```

#Filename: RandomForest.ipynb
#Author: Noah Vento
#Description: Classify data using Random Forest algorithm

#Import libraries
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import IPython.display as ipd # for display and clear_output
import time # for sleep

#Import Excel spreadsheet of data
Laguna_Fig = pd.read_excel('/Users/noahvento/Desktop/CSS_Noah/Database/M
L_Fig_Stats_Table.xlsx')

#Partition data into desired features
data = Laguna_Fig[['NTG', 'F4_Prop', 'F3_Prop', 'F2_Prop', 'F1_Prop',
                  'Gross', 'Drape', 'AR', 'Net', 'Num_Beds', 'P10_Phi_1cm',
                  'P50_Phi_1cm', 'P90_Phi_1cm', 'BT_min',
                  'BT_med', 'BT_max']]

#Make X have 16 columns and as many rows as needed to hold the values of
data
#Samples are in rows, and the features of each sample are in the columns
#The ith row of X is Sample i whose correct target output is in row i of
T
X = np.array(data).reshape((-1,16))

#Specify desired classification scheme and outputs
#(e.g., three-position classification scheme, where Axis = 1; Off-Axis =
2; Margin = 3)
Pos_3 = Laguna_Fig[['Pos_3']]

#Make T have one column and as many rows as needed to hold the values of
Pos_3
T = np.array(Pos_3).reshape((-1,1))

#Convert T outputs to 0, 1, and 2 instead of 1, 2, and 3 (Python)
def pos_conversion(self):
    T[T == 1] = 0 # convert Axis from 1 to 0
    T[T == 2] = 1 # convert Off-Axis from 2 to 1
    T[T == 3] = 2 # convert Margin from 3 to 2

pos_conversion(T)

```

```

#Separate T into Axis, Off-Axis, and Margin data
axisI, _ = np.where(T == 0) # identifying axis
offaxisI, _ = np.where(T == 1) # identifying off-axis
marginI, _ = np.where(T == 2) # identifying margin

#Randomly permute Axis, Off-Axis, and Margin data
axisI = np.random.permutation(axisI)
offaxisI = np.random.permutation(offaxisI)
marginI = np.random.permutation(marginI)

#Generate fold (K = 5) for cross-validation
length_axisI = int(len(axisI)/5) #length of each fold
axis_folds = []
for i in range(4):
    axis_folds += [axisI[i*length_axisI:(i+1)*length_axisI]]
axis_folds += [axisI[4*length_axisI:len(axisI)]]

length_offaxisI = int(len(offaxisI)/5) #length of each fold
offaxis_folds = []
for i in range(4):
    offaxis_folds += [offaxisI[i*length_offaxisI:(i+1)*length_offaxisI]]
offaxis_folds += [offaxisI[4*length_offaxisI:len(offaxisI)]]

length_margin = int(len(marginI)/5) #length of each fold
margin_folds = []
for i in range(4):
    margin_folds += [marginI[i*length_margin:(i+1)*length_margin]]
margin_folds += [marginI[4*length_margin:len(marginI)]]

#Convert lists to arrays
axis_folds = np.array(axis_folds)
offaxis_folds = np.array(offaxis_folds)
margin_folds = np.array(margin_folds)

#Partition Axis Folds
axisfold_1 = axis_folds[0]
axisfold_2 = axis_folds[1]
axisfold_3 = axis_folds[2]
axisfold_4 = axis_folds[3]
axisfold_5 = axis_folds[4]

#Partition Off-Axis Folds
offaxisfold_1 = offaxis_folds[0]
offaxisfold_2 = offaxis_folds[1]
offaxisfold_3 = offaxis_folds[2]
offaxisfold_4 = offaxis_folds[3]
offaxisfold_5 = offaxis_folds[4]

#Partition Margin Folds
marginfold_1 = margin_folds[0]
marginfold_2 = margin_folds[1]
marginfold_3 = margin_folds[2]
marginfold_4 = margin_folds[3]
marginfold_5 = margin_folds[4]

```

```

#Set up training and testing folds
rowsTrain_1 = np.hstack((axisfold_1, axisfold_2, axisfold_3, axisfold_4,
                        offaxisfold_1, offaxisfold_2, offaxisfold_3,
                        offaxisfold_4, marginfold_1, marginfold_2,
                        marginfold_3, marginfold_4))
rowsTest_1 = np.hstack((axisfold_5, offaxisfold_5, marginfold_5))

rowsTrain_2 = np.hstack((axisfold_1, axisfold_2, axisfold_3, axisfold_5,
                        offaxisfold_1, offaxisfold_2, offaxisfold_3,
                        offaxisfold_5, marginfold_1, marginfold_2,
                        marginfold_3, marginfold_5))
rowsTest_2 = np.hstack((axisfold_4, offaxisfold_4, marginfold_4))

rowsTrain_3 = np.hstack((axisfold_1, axisfold_2, axisfold_4, axisfold_5,
                        offaxisfold_1, offaxisfold_2, offaxisfold_4,
                        offaxisfold_5, marginfold_1, marginfold_2,
                        marginfold_4, marginfold_5))
rowsTest_3 = np.hstack((axisfold_3, offaxisfold_3, marginfold_3))

rowsTrain_4 = np.hstack((axisfold_1, axisfold_3, axisfold_4, axisfold_5,
                        offaxisfold_1, offaxisfold_3, offaxisfold_4,
                        offaxisfold_5, marginfold_1, marginfold_3,
                        marginfold_4, marginfold_5))
rowsTest_4 = np.hstack((axisfold_2, offaxisfold_2, marginfold_2))

rowsTrain_5 = np.hstack((axisfold_2, axisfold_3, axisfold_4, axisfold_5,
                        offaxisfold_2, offaxisfold_3, offaxisfold_4,
                        offaxisfold_5, marginfold_2, marginfold_3,
                        marginfold_4, marginfold_5))
rowsTest_5 = np.hstack((axisfold_1, offaxisfold_1, marginfold_1))

```

```

#Xtrain = Array of training features from each channel element randomly
#permuted in rowsTrain
#Ttrain = Corresponding outputs (classifications) for Xtrain values
Xtrain_1 = X[rowsTrain_1,:]
Ttrain_1 = T[rowsTrain_1,:]
Xtest_1 = X[rowsTest_1,:]
Ttest_1 = T[rowsTest_1,:]

Xtrain_2 = X[rowsTrain_2,:]
Ttrain_2 = T[rowsTrain_2,:]
Xtest_2 = X[rowsTest_2,:]
Ttest_2 = T[rowsTest_2,:]

Xtrain_3 = X[rowsTrain_3,:]
Ttrain_3 = T[rowsTrain_3,:]
Xtest_3 = X[rowsTest_3,:]
Ttest_3 = T[rowsTest_3,:]

Xtrain_4 = X[rowsTrain_4,:]
Ttrain_4 = T[rowsTrain_4,:]
Xtest_4 = X[rowsTest_4,:]
Ttest_4 = T[rowsTest_4,:]

Xtrain_5 = X[rowsTrain_5,:]
Ttrain_5 = T[rowsTrain_5,:]
Xtest_5 = X[rowsTest_5,:]
Ttest_5 = T[rowsTest_5,:]

```

```

#Define standardize function and standardize input data before classification
def standardize(X,mean,std):
    return (X - mean)/std

meanstrain_1, stdstrain_1 = np.mean(Xtrain_1, 0), np.std(Xtrain_1 ,0)
meanstest_1, stdtest_1 = np.mean(Xtest_1, 0), np.std(Xtest_1 ,0)
Xtrains_1 = standardize(Xtrain_1, meanstrain_1, stdstrain_1)
Xtests_1 = standardize(Xtest_1, meanstest_1, stdtest_1)

meanstrain_2, stdstrain_2 = np.mean(Xtrain_2, 0), np.std(Xtrain_2 ,0)
meanstest_2, stdtest_2 = np.mean(Xtest_2, 0), np.std(Xtest_2 ,0)
Xtrains_2 = standardize(Xtrain_2, meanstrain_2, stdstrain_2)
Xtests_2 = standardize(Xtest_2, meanstest_2, stdtest_2)

meanstrain_3, stdstrain_3 = np.mean(Xtrain_3, 0), np.std(Xtrain_3 ,0)
meanstest_3, stdtest_3 = np.mean(Xtest_3, 0), np.std(Xtest_3 ,0)
Xtrains_3 = standardize(Xtrain_3, meanstrain_3, stdstrain_3)
Xtests_3 = standardize(Xtest_3, meanstest_3, stdtest_3)

meanstrain_4, stdstrain_4 = np.mean(Xtrain_4, 0), np.std(Xtrain_4 ,0)
meanstest_4, stdtest_4 = np.mean(Xtest_4, 0), np.std(Xtest_4 ,0)
Xtrains_4 = standardize(Xtrain_4, meanstrain_4, stdstrain_4)
Xtests_4 = standardize(Xtest_4, meanstest_4, stdtest_4)

meanstrain_5, stdstrain_5 = np.mean(Xtrain_5, 0), np.std(Xtrain_5 ,0)
meanstest_5, stdtest_5 = np.mean(Xtest_5, 0), np.std(Xtest_5 ,0)
Xtrains_5 = standardize(Xtrain_5, meanstrain_5, stdstrain_5)
Xtests_5 = standardize(Xtest_5, meanstest_5, stdtest_5)

```

```

#Import Random Forest Classifier function from Sci-kit learn
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification

#Convert Ttrain into a flattened array
Ttrain_1 = np.ravel(Ttrain_1)
Ttrain_2 = np.ravel(Ttrain_2)
Ttrain_3 = np.ravel(Ttrain_3)
Ttrain_4 = np.ravel(Ttrain_4)
Ttrain_5 = np.ravel(Ttrain_5)

#Use Random Forest Classifier to classify all folds
clf = RandomForestClassifier()
clf_1 = clf.fit(Xtrains_1, Ttrain_1)
Ytest_1 = clf_1.predict(Xtests_1)
Ptest_1 = clf_1.predict_proba(Xtests_1)

clf_2 = clf.fit(Xtrains_2, Ttrain_2)
Ytest_2 = clf_2.predict(Xtests_2)
Ptest_2 = clf_2.predict_proba(Xtests_2)

clf_3 = clf.fit(Xtrains_3, Ttrain_3)
Ytest_3 = clf_3.predict(Xtests_3)
Ptest_3 = clf_3.predict_proba(Xtests_3)

clf_4 = clf.fit(Xtrains_4, Ttrain_4)
Ytest_4 = clf_4.predict(Xtests_4)
Ptest_4 = clf_4.predict_proba(Xtests_4)

clf_5 = clf.fit(Xtrains_5, Ttrain_5)
Ytest_5 = clf_5.predict(Xtests_5)
Ptest_5 = clf_5.predict_proba(Xtests_5)

#Reshape outputs of test folds to vectors
Ytest_1 = np.array(Ytest_1).reshape(-1,1)
Ytest_2 = np.array(Ytest_2).reshape(-1,1)
Ytest_3 = np.array(Ytest_3).reshape(-1,1)
Ytest_4 = np.array(Ytest_4).reshape(-1,1)
Ytest_5 = np.array(Ytest_5).reshape(-1,1)

#Calculate accuracies of each test fold by comparing to true labels
testacc_1 = (np.sum(Ytest_1 == Ttest_1)/len(Ttest_1))*100
testacc_2 = (np.sum(Ytest_2 == Ttest_2)/len(Ttest_2))*100
testacc_3 = (np.sum(Ytest_3 == Ttest_3)/len(Ttest_3))*100
testacc_4 = (np.sum(Ytest_4 == Ttest_4)/len(Ttest_4))*100
testacc_5 = (np.sum(Ytest_5 == Ttest_5)/len(Ttest_5))*100

#Find highest accuracy from test folds
highest_acc = max((testacc_1, testacc_2, testacc_3, testacc_4, testacc_5
))

#Concatenate all folds to calculate accuracy
Ytest_tot = np.concatenate((Ytest_1, Ytest_2, Ytest_3, Ytest_4, Ttest_5
))
Ttest_tot = np.concatenate((Ttest_1, Ttest_2, Ttest_3, Ttest_4, Ttest_5
))

#Calculate accuracy
crossval_acc = (np.sum(Ytest_tot == Ttest_tot)/len(Ttest_tot))*100

#Print cross-validation accuracy and highest accuracy
crossval_acc, highest_acc

```


XGBoost.ipynb

```
#Filename: XGBoost.ipynb
#Author: Noah Vento
#Description: Classify data using Extreme Gradient Boosting (XGBoost) algorithm

#Import libraries
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import IPython.display as ipd # for display and clear_output
import time # for sleep

#Import Excel spreadsheet of data
Laguna_Fig = pd.read_excel('/Users/noahvento/Desktop/CSS_Noah/Database/ML_Fig_Stats_Table.xlsx')

#Partition data into desired features
data = Laguna_Fig[['NTG', 'F4_Prop', 'F3_Prop', 'F2_Prop', 'F1_Prop',
                  'Gross', 'Drape', 'AR', 'Net', 'Num_Beds', 'P10_Phi_1cm',
                  'P50_Phi_1cm', 'P90_Phi_1cm', 'BT_min',
                  'BT_med', 'BT_max']]

#Make X have 16 columns and as many rows as needed to hold the values of
data
#Samples are in rows, and the features of each sample are in the columns
#The ith row of X is Sample i whose correct target output is in row i of
T
X = np.array(data).reshape((-1,16))

#Specify desired classification scheme and outputs
#(e.g., three-position classification scheme, where Axis = 1; Off-Axis =
2; Margin = 3)
Pos_3 = Laguna_Fig[['Pos_3']]

#Make T have one column and as many rows as needed to hold the values of
Pos_3
T = np.array(Pos_3).reshape((-1,1))

#Convert T outputs to 0, 1, and 2 instead of 1, 2, and 3 (Python)
def pos_conversion(self):
    T[T == 1] = 0 # convert Axis from 1 to 0
    T[T == 2] = 1 # convert Off-Axis from 2 to 1
    T[T == 3] = 2 # convert Margin from 3 to 2

pos_conversion(T)
```

```

#Separate T into Axis, Off-Axis, and Margin data
axisI, _ = np.where(T == 0) # identifying axis
offaxisI, _ = np.where(T == 1) # identifying off-axis
marginI, _ = np.where(T == 2) # identifying margin

#Randomly permute Axis, Off-Axis, and Margin data
axisI = np.random.permutation(axisI)
offaxisI = np.random.permutation(offaxisI)
marginI = np.random.permutation(marginI)

#Generate fold (K = 5) for cross-validation
length_axisI = int(len(axisI)/5) #length of each fold
axis_folds = []
for i in range(4):
    axis_folds += [axisI[i*length_axisI:(i+1)*length_axisI]]
axis_folds += [axisI[4*length_axisI:len(axisI)]]

length_offaxisI = int(len(offaxisI)/5) #length of each fold
offaxis_folds = []
for i in range(4):
    offaxis_folds += [offaxisI[i*length_offaxisI:(i+1)*length_offaxisI]]
offaxis_folds += [offaxisI[4*length_offaxisI:len(offaxisI)]]

length_margin = int(len(marginI)/5) #length of each fold
margin_folds = []
for i in range(4):
    margin_folds += [marginI[i*length_margin:(i+1)*length_margin]]
margin_folds += [marginI[4*length_margin:len(marginI)]]

#Convert lists to arrays
axis_folds = np.array(axis_folds)
offaxis_folds = np.array(offaxis_folds)
margin_folds = np.array(margin_folds)

#Partition Axis Folds
axisfold_1 = axis_folds[0]
axisfold_2 = axis_folds[1]
axisfold_3 = axis_folds[2]
axisfold_4 = axis_folds[3]
axisfold_5 = axis_folds[4]

#Partition Off-Axis Folds
offaxisfold_1 = offaxis_folds[0]
offaxisfold_2 = offaxis_folds[1]
offaxisfold_3 = offaxis_folds[2]
offaxisfold_4 = offaxis_folds[3]
offaxisfold_5 = offaxis_folds[4]

#Partition Margin Folds
marginfold_1 = margin_folds[0]
marginfold_2 = margin_folds[1]
marginfold_3 = margin_folds[2]
marginfold_4 = margin_folds[3]
marginfold_5 = margin_folds[4]

```

```

#Set up training and testing folds
rowsTrain_1 = np.hstack((axisfold_1, axisfold_2, axisfold_3, axisfold_4,
                        offaxisfold_1, offaxisfold_2, offaxisfold_3,
                        offaxisfold_4, marginfold_1, marginfold_2,
                        marginfold_3, marginfold_4))
rowsTest_1 = np.hstack((axisfold_5, offaxisfold_5, marginfold_5))

rowsTrain_2 = np.hstack((axisfold_1, axisfold_2, axisfold_3, axisfold_5,
                        offaxisfold_1, offaxisfold_2, offaxisfold_3,
                        offaxisfold_5, marginfold_1, marginfold_2,
                        marginfold_3, marginfold_5))
rowsTest_2 = np.hstack((axisfold_4, offaxisfold_4, marginfold_4))

rowsTrain_3 = np.hstack((axisfold_1, axisfold_2, axisfold_4, axisfold_5,
                        offaxisfold_1, offaxisfold_2, offaxisfold_4,
                        offaxisfold_5, marginfold_1, marginfold_2,
                        marginfold_4, marginfold_5))
rowsTest_3 = np.hstack((axisfold_3, offaxisfold_3, marginfold_3))

rowsTrain_4 = np.hstack((axisfold_1, axisfold_3, axisfold_4, axisfold_5,
                        offaxisfold_1, offaxisfold_3, offaxisfold_4,
                        offaxisfold_5, marginfold_1, marginfold_3,
                        marginfold_4, marginfold_5))
rowsTest_4 = np.hstack((axisfold_2, offaxisfold_2, marginfold_2))

rowsTrain_5 = np.hstack((axisfold_2, axisfold_3, axisfold_4, axisfold_5,
                        offaxisfold_2, offaxisfold_3, offaxisfold_4,
                        offaxisfold_5, marginfold_2, marginfold_3,
                        marginfold_4, marginfold_5))
rowsTest_5 = np.hstack((axisfold_1, offaxisfold_1, marginfold_1))

```

```

#Xtrain = Array of training features from each channel element randomly
#permuted in rowsTrain
#Ttrain = Corresponding outputs (classifications) for Xtrain values
Xtrain_1 = X[rowsTrain_1,:]
Ttrain_1 = T[rowsTrain_1,:]
Xtest_1 = X[rowsTest_1,:]
Ttest_1 = T[rowsTest_1,:]

Xtrain_2 = X[rowsTrain_2,:]
Ttrain_2 = T[rowsTrain_2,:]
Xtest_2 = X[rowsTest_2,:]
Ttest_2 = T[rowsTest_2,:]

Xtrain_3 = X[rowsTrain_3,:]
Ttrain_3 = T[rowsTrain_3,:]
Xtest_3 = X[rowsTest_3,:]
Ttest_3 = T[rowsTest_3,:]

Xtrain_4 = X[rowsTrain_4,:]
Ttrain_4 = T[rowsTrain_4,:]
Xtest_4 = X[rowsTest_4,:]
Ttest_4 = T[rowsTest_4,:]

Xtrain_5 = X[rowsTrain_5,:]
Ttrain_5 = T[rowsTrain_5,:]
Xtest_5 = X[rowsTest_5,:]
Ttest_5 = T[rowsTest_5,:]

```

```

#Define standardize function and standardize input data before classification
def standardize(X,mean,stds):
    return (X - mean)/stds

meanstrain_1, stdstrain_1 = np.mean(Xtrain_1, 0), np.std(Xtrain_1 ,0)
meanstest_1, stdtest_1 = np.mean(Xtest_1, 0), np.std(Xtest_1 ,0)
Xtrains_1 = standardize(Xtrain_1, meanstrain_1, stdstrain_1)
Xtests_1 = standardize(Xtest_1, meanstest_1, stdtest_1)

meanstrain_2, stdstrain_2 = np.mean(Xtrain_2, 0), np.std(Xtrain_2 ,0)
meanstest_2, stdtest_2 = np.mean(Xtest_2, 0), np.std(Xtest_2 ,0)
Xtrains_2 = standardize(Xtrain_2, meanstrain_2, stdstrain_2)
Xtests_2 = standardize(Xtest_2, meanstest_2, stdtest_2)

meanstrain_3, stdstrain_3 = np.mean(Xtrain_3, 0), np.std(Xtrain_3 ,0)
meanstest_3, stdtest_3 = np.mean(Xtest_3, 0), np.std(Xtest_3 ,0)
Xtrains_3 = standardize(Xtrain_3, meanstrain_3, stdstrain_3)
Xtests_3 = standardize(Xtest_3, meanstest_3, stdtest_3)

meanstrain_4, stdstrain_4 = np.mean(Xtrain_4, 0), np.std(Xtrain_4 ,0)
meanstest_4, stdtest_4 = np.mean(Xtest_4, 0), np.std(Xtest_4 ,0)
Xtrains_4 = standardize(Xtrain_4, meanstrain_4, stdstrain_4)
Xtests_4 = standardize(Xtest_4, meanstest_4, stdtest_4)

meanstrain_5, stdstrain_5 = np.mean(Xtrain_5, 0), np.std(Xtrain_5 ,0)
meanstest_5, stdtest_5 = np.mean(Xtest_5, 0), np.std(Xtest_5 ,0)
Xtrains_5 = standardize(Xtrain_5, meanstrain_5, stdstrain_5)
Xtests_5 = standardize(Xtest_5, meanstest_5, stdtest_5)

```

```

1  #Import XGBoost function
2  import xgboost as xgb
3
4  #Convert Ttrain into a flattened array
5  Ttrain_1 = np.ravel(Ttrain_1)
6  Ttrain_2 = np.ravel(Ttrain_2)
7  Ttrain_3 = np.ravel(Ttrain_3)
8  Ttrain_4 = np.ravel(Ttrain_4)
9  Ttrain_5 = np.ravel(Ttrain_5)
10
11 #Use XGBoost to classify all folds
12 #Ptest = Probabilities for classifications
13 clf = xgb.XGBClassifier()
14 clf_1 = clf.fit(Xtrains_1, Ttrain_1)
15 Ytest_1 = clf_1.predict(Xtests_1)
16 Ptest_1 = clf_1.predict_proba(Xtests_1)
17
18 clf_2 = clf.fit(Xtrains_2, Ttrain_2)
19 Ytest_2 = clf_2.predict(Xtests_2)
20 Ptest_2 = clf_2.predict_proba(Xtests_2)
21
22 clf_3 = clf.fit(Xtrains_3, Ttrain_3)
23 Ytest_3 = clf_3.predict(Xtests_3)
24 Ptest_3 = clf_3.predict_proba(Xtests_3)
25
26 clf_4 = clf.fit(Xtrains_4, Ttrain_4)
27 Ytest_4 = clf_4.predict(Xtests_4)
28 Ptest_4 = clf_4.predict_proba(Xtests_4)
29
30 clf_5 = clf.fit(Xtrains_5, Ttrain_5)
31 Ytest_5 = clf_5.predict(Xtests_5)
32 Ptest_5 = clf_5.predict_proba(Xtests_5)

```

```

1  #Reshape outputs of test folds to vectors
2  Ytest_1 = np.array(Ytest_1).reshape(-1,1)
3  Ytest_2 = np.array(Ytest_2).reshape(-1,1)
4  Ytest_3 = np.array(Ytest_3).reshape(-1,1)
5  Ytest_4 = np.array(Ytest_4).reshape(-1,1)
6  Ytest_5 = np.array(Ytest_5).reshape(-1,1)

```

```

#Calculate accuracies of each test fold by comparing to true labels
testacc_1 = (np.sum(Ytest_1 == Ttest_1)/len(Ttest_1))*100
testacc_2 = (np.sum(Ytest_2 == Ttest_2)/len(Ttest_2))*100
testacc_3 = (np.sum(Ytest_3 == Ttest_3)/len(Ttest_3))*100
testacc_4 = (np.sum(Ytest_4 == Ttest_4)/len(Ttest_4))*100
testacc_5 = (np.sum(Ytest_5 == Ttest_5)/len(Ttest_5))*100

#Find highest accuracy from test folds
highest_acc = max((testacc_1, testacc_2, testacc_3, testacc_4, testacc_5
))

#Concatenate all folds to calculate accuracy
Ytest_tot = np.concatenate((Ytest_1, Ytest_2, Ytest_3, Ytest_4, Ttest_5
))
Ttest_tot = np.concatenate((Ttest_1, Ttest_2, Ttest_3, Ttest_4, Ttest_5
))
Ptest_tot = np.concatenate((Ptest_1, Ptest_2, Ptest_3, Ptest_4, Ptest_5
))

#Calculate accuracy
crossval_acc = (np.sum(Ytest_tot == Ttest_tot)/len(Ttest_tot))*100

#Print cross-validation accuracy and highest accuracy
crossval_acc, highest_acc

```

optimizers.py

```
#Filename: optimizers.py
#Author: Charles Anderson
#Description: The optimizers script defines different optimization algorithms
 (e.g., Stochastic Gradient Descent, Adam, and Scaled Conjugate Gradient
 Descent).
#These optimizers are used during the backpropagation phase of the Neural
 Network
#to update the weights and biases and minimize the loss function.

import numpy as np
import sys
import copy
import time
import math # for math.ceil
import torch

#####
## sgd

def sgd(w, error_f, fargs=[], n_iterations=100, error_gradient_f=None,
        eval_f=lambda x: x, save_wtrace=False, verbose=False,
        use_torch=False,
        learning_rate=0.001, momentum_rate=0.0):

    start_time = time.time()
    start_time_last_verbose = start_time

    if use_torch:
        if isinstance(w, np.ndarray):
            w = torch.tensor(w, dtype=torch.float, requires_grad=True)
        else:
            w = w.clone().detach().requires_grad_(True)
        wtrace = [w.clone().detach()] if save_wtrace else None
    else:
        w = w.copy()
        wtrace = [w.copy()] if save_wtrace else None

    ftrace = [eval_f(error_f(w, *fargs))]

    w_change = 0

    for iteration in range(n_iterations):

        error = error_f(w, *fargs)

        if use_torch:
            error.backward(retain_graph=True)
            with torch.no_grad():
                w_change = -learning_rate * w.grad + momentum_rate * w_c
change
                w += w_change
                w.grad.zero_()
            if save_wtrace:
                wtrace.append(w.clone().detach())
                ftrace.append(eval_f(error).detach())
```

```

else:
    grad = error_gradient_f(w, *fargs)
    w_change = -learning_rate * grad + momentum_rate * w_change
    w += w_change
    if save_wtrace:
        wtrace.append(w.copy())
        ftrace.append(eval_f(error))

iterations_per_print = math.ceil(n_iterations/10)
if verbose and (iteration + 1) % max(1, iterations_per_print) ==
0:
    seconds = time.time() - start_time_last_verbose
    eval = eval_f(error) # .item() if use_torch else eval_f(error)
    print(f'sgd: Iteration {iteration+1:d} ObjectiveF={eval:.5f}
Seconds={seconds:.3f}')
    start_time_last_verbose = time.time()

return {'w': w,
        'f': error_f(w, *fargs),
        'n_iterations': iteration,
        'wtrace': np.array(wtrace)[:iteration + 2,:] if save_wtrace
else None,
        'ftrace': np.array(ftrace)[:iteration + 2],
        'reason': 'iterations',
        'time': time.time() - start_time}

#####
## adam

def adam(w, error_f, fargs=[], n_iterations=100, error_gradient_f=None,
eval_f=lambda x: x, save_wtrace=False, verbose=False,
use_torch=False,
learning_rate=0.001, momentum_rate=None):

start_time = time.time()
start_time_last_verbose = start_time

if use_torch:
    if isinstance(w, np.ndarray):
        w = torch.tensor(w, dtype=torch.float, requires_grad=True)
    else:
        w = w.clone().detach().requires_grad_(True)
    wtrace = [w.clone()] if save_wtrace else None
else:
    w = w.copy()
    wtrace = [w.copy()] if save_wtrace else None

beta1 = 0.9
beta2 = 0.999
alpha = learning_rate
epsilon = 10e-8
nW = len(w)
g = torch.zeros((nW)) if use_torch else np.zeros((nW))

```

```

g2 = torch.zeros((nW)) if use_torch else np.zeros((nW))
betalt = beta1
beta2t = beta2

ftrace = [eval_f(error_f(w, *fargs))]

for iteration in range(n_iterations):

    error = error_f(w, *fargs)

    if use_torch:
        error.backward(retain_graph=True)
        with torch.no_grad():
            g = beta1 * g + (1 - beta1) * w.grad
            g2 = beta2 * g2 + (1 - beta2) * w.grad * w.grad
            g_corrected = g / (1 - betalt)
            g2_corrected = g2 / (1 - beta2t)
            # alphas = alpha * torch.sqrt(1 - beta2t) / (1 - betalt)
            w -= alpha * g_corrected / (torch.sqrt(g2_corrected) + epsilon)

            w.grad.zero_()

        if save_wtrace:
            wtrace.append(w.clone().detach())
            ftrace.append(eval_f(error).detach())

    else:
        grad = error_gradient_f(w, *fargs)
        g = beta1 * g + (1 - beta1) * grad
        g2 = beta2 * g2 + (1 - beta2) * grad * grad
        g_corrected = g / (1 - betalt)
        g2_corrected = g2 / (1 - beta2t)
        w -= alpha * g_corrected / (np.sqrt(g2_corrected) + epsilon)
        if save_wtrace:
            wtrace.append(w.copy())
            ftrace.append(eval_f(error))

    betalt *= beta1
    beta2t *= beta2

    iterations_per_print = math.ceil(n_iterations/10)
    if verbose and (iteration + 1) % max(1, iterations_per_print) ==
0:
        seconds = time.time() - start_time_last_verbose
        eval = eval_f(error) # .item() if use_torch else eval_f(error)
        print(f'adam: Iteration {iteration+1:d} ObjectiveF={eval:.5
f} Seconds={seconds:.3f}')
        start_time_last_verbose = time.time()

    return {'w': w,
            'f': error_f(w, *fargs),
            'n_iterations': iteration,
            'wtrace': np.array(wtrace)[:iteration + 2, :] if save_wtrace
else None,
            'ftrace': np.array(ftrace)[:iteration + 2, :]

```



```

        'reason': 'iterations',
        'time': time.time() - start_time}

#####
# Scaled Conjugate Gradient algorithm from
# "A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning"
# by Martin F. Moller
# Neural Networks, vol. 6, pp. 525-533, 1993
#
# Adapted by Chuck Anderson from the Matlab implementation by Nabney
# as part of the netlab library.
#

def scg(w, error_f, fargs=[], n_iterations=100, error_gradient_f=None,
        eval_f=lambda x: x, save_wtrace=False, verbose=False,
        use_torch=False,
        learning_rate=None, momentum_rate=None): # not used here

    float_precision = sys.float_info.epsilon

    start_time = time.time()
    start_time_last_verbose = start_time

    if use_torch:
        if isinstance(w, np.ndarray):
            w = torch.tensor(w, dtype=torch.float, requires_grad=True)
        else:
            w = w.clone().detach().requires_grad_(True)
            wtrace = [w.clone()] if save_wtrace else None
            isnan = torch.isnan
            sqrt = torch.sqrt
    else:
        w = w.copy()
        wtrace = [w.copy()] if save_wtrace else None
        isnan = np.isnan
        sqrt = math.sqrt

    sigma0 = 1.0e-6
    error_old = error_f(w, *fargs)
    error_now = error_old
    if use_torch:
        error_now.backward(retain_graph=True)
        gradnew = w.grad.clone()
        w.grad.zero_()
        ftrace = [eval_f(error_old).detach()]
    else:
        gradnew = error_gradient_f(w, *fargs)
        ftrace = [eval_f(error_old)]

    gradold = copy.deepcopy(gradnew)
    d = -gradnew # Initial search direction.
    success = True # Force calculation of directional derivs.
    nsuccess = 0 # nsuccess counts number of successes.
    beta = 1.0e-6 # Initial scale parameter. Lambda in Moeller.
    betamin = 1.0e-15 # Lower bound on scale.
    betamax = 1.0e20 # Upper bound on scale.
    nvars = len(w)

```

```

iteration = 1      # count of number of iterations

thisIteration = 1

while thisIteration <= n_iterations:

    if success:
        mu = d.T @ gradnew
        if mu >= 0:
            d = -gradnew
            mu = d.T @ gradnew
            kappa = d.T @ d

            if isnan(kappa):
                print('kappa', kappa)

            if kappa < float_precision:
                return {'w': w,
                        'f': error_now,
                        'n_iterations': iteration,
                        'wtrace': np.array(wtrace)[:iteration + 1, :] if
save_wtrace else None,
                        'ftrace': np.array(ftrace)[:iteration + 1],
                        'reason': 'limit on machine precision',
                        'time': time.time() - start_time}
            sigma = sigma0 / sqrt(kappa)

            if use_torch:
                w_smallstep = (w.detach() + sigma * d).requires_grad_(True)

                err = error_f(w_smallstep, *fargs)
                err.backward(retain_graph=True)
                g_smallstep = w_smallstep.grad.clone()
                w_smallstep.grad.zero_()
            else:
                w_smallstep = w + sigma * d
                error_f(w_smallstep, *fargs)
                g_smallstep = error_gradient_f(w_smallstep, *fargs)

            theta = d.T @ (g_smallstep - gradnew) / sigma
            if isnan(theta):
                print(f'theta {theta} sigma {sigma} d[0] {d[0]} g_smallstep[0] {g_smallstep[0]} gradnew[0] {gradnew[0]}')

            ## Increase effective curvature and evaluate step size alpha.
            delta = theta + beta * kappa
            if isnan(delta):
                print(f'delta is NaN theta {theta} beta {beta} kappa {kappa}')

            elif delta <= 0:
                delta = beta * kappa
                beta = beta - theta / kappa

            if delta == 0:
                success = False
                error_now = error_old

```

```

else:
    alpha = -mu / delta
    ## Calculate the comparison ratio Delta
    if use_torch:
        wnew = (w.detach() + alpha * d).requires_grad_(True)
    else:
        wnew = w + alpha * d
    error_new = error_f(wnew, *fargs)
    Delta = 2 * (error_new - error_old) / (alpha * mu)
    if not isnan(Delta) and Delta >= 0:
        success = True
        nsuccess += 1
        if use_torch:
            w = wnew.detach().requires_grad_(True)
        else:
            w[:] = wnew
        error_now = error_new
    else:
        success = False
        error_now = error_old

iterations_per_print = math.ceil(n_iterations/10)
if verbose and thisIteration % max(1, iterations_per_print) == 0
:
    seconds = time.time() - start_time_last_verbose
    print(f'SCG: Iteration {iteration:d} ObjectiveF={eval_f(error
r_now):.5f} Scale={beta:.3e} Seconds={seconds:.3f}')
    start_time_last_verbose = time.time()
    if use_torch:
        if save_wtrace:
            wtrace.append(w.clone().detach())
            ftrace.append(eval_f(error_now).detach())
    else:
        if save_wtrace:
            wtrace.append(w.copy())
            ftrace.append(eval_f(error_now))

if success:

    error_old = error_new
    gradold[:] = gradnew
    if use_torch:
        error_new.backward(retain_graph=True)
        gradnew = wnew.grad.clone()
        wnew.grad.zero_()
    else:
        gradnew[:] = error_gradient_f(w, *fargs)

# If the gradient is zero then we are done.
    gg = gradnew.T @ gradnew
    if gg == 0:
        return {'w': w,
                'f': error_now,
                'n_iterations': iteration,
                'wtrace': np.array(wtrace)[:iteration + 1, :]} if
save_wtrace else None,

```

```

        'ftrace': np.array(ftrace)[:iteration + 1],
        'reason': 'zero gradient',
        'time': time.time() - start_time}

    if isnan(Delta) or Delta < 0.25:
        beta = min(4.0 * beta, betamax)
    elif Delta > 0.75:
        beta = max(0.5 * beta, betamin)

    # Update search direction using Polak-Ribiere formula, or re-start
    # in direction of negative gradient after nparams steps.
    if nsuccess == nvars:
        d[:] = -gradnew
        nsuccess = 0
    elif success:
        gamma = (gradold - gradnew).T @ (gradnew / mu)
        d[:] = gamma * d - gradnew

    thisIteration += 1
    iteration += 1

    # If we get here, then we haven't terminated in the given number
    of iterations.

    return {'w': w,
            'f': error_now,
            'n_iterations': iteration,
            'wtrace': np.array(wtrace)[:iteration + 1, :] if save_wtrace
else None,
            'ftrace': np.array(ftrace)[:iteration + 1],
            'reason': 'did not converge',
            'time': time.time() - start_time}

if __name__ == '__main__':

    def error(w):
        return (w - 1.5)**2

    def error_grad(w):
        return 2 * (w - 1.5)

    w = np.array([-5.5])

    result = sgd(w, error, [], 1000, error_grad, use_torch=False,
                 learning_rate=0.1, momentum_rate=0.5)
    print(f"sgd w is {result['w'][0]:.3f}")

    result = adam(w, error, [], 1000, error_grad, use_torch=False,
                  learning_rate=0.1)
    print(f"adam w is {result['w'][0]:.3f}")

    result = scg(w, error, [], 1000, error_grad, use_torch=False)
    print(f"scg w is {result['w'][0]:.3f}")

    result = sgd(w, error, [], 1000, error_grad, use_torch=True,

```

```

        learning_rate=0.1, momentum_rate=0.5)
print(f"sgd with torch, w is {result['w'][0]:.3f}")

result = adam(w, error, [], 1000, error_grad, use_torch=True,
              learning_rate=0.1)
print(f"adam with torch, w is {result['w'][0]:.3f}")

result = scg(w, error, [], 1000, error_grad, use_torch=True)
print(f"scg with torch, w is {result['w'][0]:.3f}")

```

neuralnetworks.py

```

#Filename: neuralnetworks.py
#Author: Charles Anderson
#Description: The neuralnetworks script defines the architecture
#and components of the neural network.

import numpy as np
import torch
import mlutilities as ml
import optimizers as opt
import matplotlib.pyplot as plt
import copy

class NeuralNetwork:

    #Defining the operator
    def __init__(self, n_inputs, n_hidden_list, n_outputs, use_torch=False):

        if not isinstance(n_hidden_list, list):
            raise Exception('NeuralNetwork: n_hidden_list must be a list.')

        if len(n_hidden_list) == 0:
            self.n_hidden_layers = 0
        elif n_hidden_list[0] == 0:
            self.n_hidden_layers = 0
        else:
            self.n_hidden_layers = len(n_hidden_list)

        self.n_inputs = n_inputs
        self.n_hidden_list = n_hidden_list
        self.n_outputs = n_outputs

        # Do we have any hidden layers?
        self.Vs = []
        ni = n_inputs
        for layeri in range(self.n_hidden_layers):
            n_in_layer = self.n_hidden_list[layeri]
            self.Vs.append(1 / np.sqrt(1 + ni) * np.random.uniform(-1, 1,
                size=(1 + ni, n_in_layer)))
            ni = n_in_layer
            self.W = 1/np.sqrt(1 + ni) * np.random.uniform(-1, 1, size=(1 +
                ni, n_outputs))
            if use_torch:
                self.Vs = [torch.tensor(V, dtype=torch.float) for V in self.
                Vs]

                self.W = torch.tensor(self.W, dtype=torch.float)
                self.tanh = torch.tanh
                self.mean = torch.mean
                self.sqrt = torch.sqrt
            else:
                self.tanh = np.tanh
                self.mean = np.mean
                self.sqrt = np.sqrt

```

```

self.use_torch = use_torch

# Member variables for standardization
self.Xmeans = None
self.Xstds = None
self.Tmeans = None
self.Tstds = None

self.trained = False
self.reason = None
self.error_trace = None
self.n_epochs = None
self.training_time = None

#Representation for the neural network
def __repr__(self):
    str = f'{type(self).__name__}({self.n_inputs}, {self.n_hidden_l
ist}, {self.n_outputs}, use_torch={self.use_torch})'
    if self.trained:
        str += f'\n  Network was trained for {self.n_epochs} epoch
s'
        str += f' that took {self.training_time:.4f} seconds. Final
objective value is {self.error_trace[-1]:.3f}'
    else:
        str += '  Network is not trained.'
    return str

#Standardizing inputs -- standard approach in Machine Learning
def _standardizeX(self, X):
    result = (X - self.Xmeans) / self.XstdsFixed
    result[:, self.Xconstant] = 0.0
    return result

def _unstandardizeX(self, Xs):
    return self.Xstds * Xs + self.Xmeans

def _standardizeT(self, T):
    result = (T - self.Tmeans) / self.TstdsFixed
    result[:, self.Tconstant] = 0.0
    return result

def _unstandardizeT(self, Ts):
    return self.Tstds * Ts + self.Tmeans

#Packing weights into vector
def _pack(self, Vs, W):
    if self.use_torch:
        return torch.cat([V.reshape(-1) for V in Vs] + [W.reshape(-1
))]
    else:
        return np.hstack([V.flat for V in Vs] + [W.flat])

def _unpack(self, w):
    first = 0
    n_this_layer = self.n_inputs
    for i in range(self.n_hidden_layers):
        self.Vs[i][:] = w[first:first + (1 + n_this_layer) *

```

```

        self.n_hiddens_list[i]).reshape((1 + n_this
s_layer, self.n_hiddens_list[i]))
        first += (1 + n_this_layer) * self.n_hiddens_list[i]
        n_this_layer = self.n_hiddens_list[i]
        self.W[:] = w[first:].reshape((1 + n_this_layer, self.n_outputs
))

    def _forward_pass(self, X):
        # Assume weights already unpacked
        Z_prev = X # output of previous layer
        Z = [Z_prev]
        for i in range(self.n_hidden_layers):
            V = self.Vs[i]
            Z_prev = self.tanh(Z_prev @ V[1:, :] + V[0:1, :]) #Hyperboli
c tangent function
            Z.append(Z_prev)
        Y = Z_prev @ self.W[1:, :] + self.W[0:1, :] #Y = outputs
        return Y, Z

    def _objectiveF(self, w, X, T):
        self._unpack(w)
        Y, _ = self._forward_pass(X)
        return 0.5 * self.mean((T - Y)**2) #Loss Function = Mean Square

Error

# Only used if use_torch=False
    def _gradientF(self, w, X, T):
        self._unpack(w)
        Y, Z = self._forward_pass(X)
        # Do backward pass, starting with delta in output layer
        delta = -(T - Y) / (X.shape[0] * T.shape[1])
        # Another way to define dEdW without calling np.insert
        dW = np.vstack((np.sum(delta, axis=0), Z[-1].T @ delta))
        dVs = []
        delta = (1 - Z[-1]**2) * (delta @ self.W[1:, :].T)
        for Zi in range(self.n_hidden_layers, 0, -1):
            Vi = Zi - 1 # because X is first element of Z
            dV = np.vstack((np.sum(delta, axis=0), Z[Zi-1].T @ delta))
            dVs.insert(0, dV) # like append, but at front of list of dV

s
            delta = (delta @ self.Vs[Vi][1:, :].T) * (1 - Z[Zi-1]**2)
        return self._pack(dVs, dW)

    def _setup_standardize(self, X, T):
        if self.Xmeans is None:
            self.Xmeans = X.mean(axis=0)
            self.Xstds = X.std(axis=0)
            self.Xconstant = self.Xstds == 0
            self.XstdsFixed = copy.copy(self.Xstds)
            self.XstdsFixed[self.Xconstant] = 1

        if self.Tmeans is None:
            self.Tmeans = T.mean(axis=0)
            self.Tstds = T.std(axis=0)
            self.Tconstant = self.Tstds == 0
            self.TstdsFixed = copy.copy(self.Tstds)
            self.TstdsFixed[self.Tconstant] = 1

```

```

def _objective_to_actual(self, objective):
    return self.sqrt(objective)

def train(self, X, T, n_epochs, method='scg',
          verbose=False, save_weights_history=False,
          learning_rate=0.001, momentum_rate=0.0): # only for sgd and adam

    if X.shape[1] != self.n_inputs:
        raise Exception(f'train: number of columns in X ({X.shape[1]}) not equal to number of network inputs ({self.n_inputs})')

    if self.use_torch:
        X = torch.tensor(X, dtype=torch.float) # 32 bit
        T = torch.tensor(T, dtype=torch.float)

    self._setup_standardize(X, T)
    X = self._standardizeX(X)
    T = self._standardizeT(T)

    try:
        algo = [opt.sgd, opt.adam, opt.scg][['sgd', 'adam', 'scg'].index(method)]
    except:
        raise Exception("train: method={method} not one of 'scg', 'sgd' or 'adam'")

    result = algo(self._pack(self.Vs, self.W),
                 self._objectiveF,
                 [X, T], n_epochs,
                 self._gradientF, # not used if scg
                 eval_f=self._objective_to_actual,
                 learning_rate=learning_rate, momentum_rate=momentum_rate,
                 verbose=verbose, use_torch=self.use_torch,
                 save_wtrace=save_weights_history)

    self._unpack(result['w'])
    self.reason = result['reason']
    self.error_trace = result['ftrace'] # * self.Tstds # to _unstandardize the MSEs
    self.n_epochs = len(self.error_trace) - 1
    self.trained = True
    self.weight_history = result['wtrace'] if save_weights_history else None
    self.training_time = result['time']
    return self

def use(self, X, all_outputs=False):
    if self.use_torch:
        if not isinstance(X, torch.Tensor):
            X = torch.tensor(X, dtype=torch.float)
    X = self._standardizeX(X)
    Y, Z = self._forward_pass(X)
    Y = self._unstandardizeT(Y)
    if self.use_torch:

```



```

        Y = Y.detach().cpu().numpy()
        Z = [Zi.detach().cpu().numpy() for Zi in Z]
        return (Y, Z[1:]) if all_outputs else Y

    def get_n_epochs(self):
        return self.n_epochs

    def get_error_trace(self):
        return self.error_trace

    def get_training_time(self):
        return self.training_time

    def get_weight_history(self):
        return self.weight_history

    def draw(self, input_names=None, output_names=None, gray=False):
        if self.use_torch:
            Vs = [V.detach().cpu().numpy() for V in self.Vs]
            W = self.W.detach().cpu().numpy()
        else:
            Vs = self.Vs
            W = self.W
        ml.draw(Vs, W, input_names, output_names, gray)

import numpy as np
import torch
import mlutilities as ml
import optimizers as opt
import matplotlib.pyplot as plt
import copy
import sys

class NeuralNetworkClassifier(NeuralNetwork):

    def __init__(self, n_inputs, n_hidden_list, classes, use_torch=False):
        if not isinstance(n_hidden_list, list):
            raise Exception('NeuralNetworkClassifier: n_hidden_list must be a list or tuple')
        super().__init__(n_inputs, n_hidden_list, len(classes), use_torch)

        self.classes = np.array(classes) # to allow argmax in use()
        if use_torch:
            self.log = torch.log
            self.exp = torch.exp
        else:
            self.log = np.log
            self.exp = np.exp

    def __repr__(self):
        str = f'{type(self).__name__}({self.n_inputs}, {self.n_hidden_list}, {self.n_outputs}, use_torch={self.use_torch})'

```

```

        if self.trained:
            str += f'\n    Network was trained for {self.n_epochs} epoch
s'
            str += f' that took {self.training_time:.4f} seconds. Final
objective value is {self.error_trace[-1]:.3f}'
        else:
            str += '    Network is not trained.'
            return str

        str = f'{type(self).__name__}({self.n_inputs}, {self.n_hidden_l
ist}, {self.classes}, use_torch={self.use_torch})'
        if self.trained:
            str += f'\n    Network was trained for {self.n_epochs} epoch
s'
            str += f' that took {self.training_time:.4f} seconds. Final
objective value is {self.error_trace[-1]:.3f}'
        else:
            str += '    Network is not trained.'
            return str

    def _standardizeT(self, T):
        return T

    def _unstandardizeT(self, Ts):
        return Ts

    def _softmax(self, Y):
        mx = Y.max()
        expY = self.exp(Y - mx)
        denom = expY.sum(axis=1).reshape((-1, 1)) + sys.float_info.epsilon
on
        return expY / denom

    def _softmax_old(self, Y):
        expY = self.exp(Y)
        denom = expY.sum(axis=1).reshape((-1, 1))
        return expY / denom

    def _forward_pass(self, X):
        Y, Z = super()._forward_pass(X)
        # Convert outputs to multinomial distribution
        return self._softmax(Y), Z

    def _objectiveF(self, w, X, T):
        self.unpack(w)
        Y, Z = self._forward_pass(X)
        Y[Y == 0] = sys.float_info.epsilon # to avoid log(0) below
        return -(T * self.log(Y)).mean()

    # _gradientF of parent works for classifier!

    def _make_indicator_variables(self, T):
        ''' Assumes argument is N x 1, N samples each being integer clas
s label '''
        return (T == np.unique(T)).astype(int)

    def _objective_to_actual(self, objective):

```

```

        return self.exp(-objective)

    def train(self, X, T, n_epochs, method='scg',
              verbose=False, save_weights_history=False,
              learning_rate=0.001, momentum_rate=0.0): # only for sgd and adam

        Ti = self._make_indicator_variables(T)
        return super().train(X, Ti, n_epochs, method, verbose, save_weights_history,
                             learning_rate, momentum_rate)

    def use(self, X, all_outputs=False):
        Y, Z = super().use(X, all_outputs=True)
        # Convert max Y to class label
        Y_classes = self.classes[Y.argmax(axis=1)].reshape((-1, 1))
        # Y_classes = Y.argmax(axis=1).reshape((-1, 1))
        return (Y_classes, Y, Z[1:]) if all_outputs else (Y_classes, Y)

if __name__ == '__main__':

    n_samples = 20
    X = np.random.choice(3, (n_samples, 2))
    T = (X[:, 0:1] == X[:, 1:2]).astype(int) # where the two inputs are equal
    classes = [0, 1]
    print(f'{np.sum(T==0)} not equal, {np.sum(T==1)} equal')

    n_epochs = 100

    for use_torch in [False, True]:

        nnet = NeuralNetworkClassifier(2, [], classes, use_torch=use_torch)

        nnet.train(X, T, n_epochs)
        Y_classes, Y = nnet.use(X)
        print(f'scg {nnet.n_hidden_list} use_torch={use_torch} {np.sum(Y_classes == T)} / {n_samples} correct took {nnet.training_time:.3f} seconds')

        nnet = NeuralNetworkClassifier(2, [20, 5], classes, use_torch=use_torch)

        nnet.train(X, T, n_epochs)
        Y_classes, Y = nnet.use(X)
        print(f'scg {nnet.n_hidden_list} use_torch={use_torch} {np.sum(Y_classes == T)} / {n_samples} correct took {nnet.training_time:.3f} seconds')

        nnet = NeuralNetworkClassifier(2, [20, 5], classes, use_torch=use_torch)

        nnet.train(X, T, n_epochs, method='sgd', learning_rate=0.5, momentum_rate=0.5)
        Y_classes, Y = nnet.use(X)
        print(f'sgd {nnet.n_hidden_list} use_torch={use_torch} {np.sum(Y_classes == T)} / {n_samples} correct took {nnet.training_time:.3f} seconds')

```

```

        nnet = NeuralNetworkClassifier(2, [20, 5], classes, use_torch=use_torch)
        nnet.train(X, T, n_epochs, method='adam', learning_rate=0.1)
        Y_classes, Y = nnet.use(X)
        print(f'adam {nnet.n_hidden_list} use_torch={use_torch} {np.sum
(Y_classes == T)} / {n_samples} correct took {nnet.training_time:.3f} seconds')

plt.figure(1)
plt.clf()
nnet.draw()

if __name__ == '__main__':

    np.random.seed(42)
    print('Called np.random.seed(42)')

    X = np.arange(10).reshape((-1, 1))
    T = X ** 2
    n_epochs = 200

    def rmse(Y, T):
        return np.sqrt(np.mean((T - Y)**2))

    for use_torch in [False, True]:

        nnet = NeuralNetwork(1, [], 1, use_torch=use_torch)
        # Equivalent to
        # nnet = NeuralNetwork(1, [0], 1, use_torch=use_torch)
        nnet.train(X, T, n_epochs)
        Y = nnet.use(X)
        print(f'scg {nnet.n_hidden_list} use_torch={use_torch} RMSE {rmse(Y, T):.3f} took {nnet.training_time:.3f} seconds')

        nnet = NeuralNetwork(1, [5, 5], 1, use_torch=use_torch)
        nnet.train(X, T, n_epochs)
        Y = nnet.use(X)
        print(f'scg {nnet.n_hidden_list} use_torch={use_torch} RMSE {rmse(Y, T):.3f} took {nnet.training_time:.3f} seconds')

        nnet = NeuralNetwork(1, [5, 5], 1, use_torch=use_torch)
        nnet.train(X, T, n_epochs, method='sgd', learning_rate=0.5, momentum_rate=0.5)
        Y = nnet.use(X)
        print(f'sgd {nnet.n_hidden_list} use_torch={use_torch} RMSE {rmse(Y, T):.3f} took {nnet.training_time:.3f} seconds')

        nnet = NeuralNetwork(1, [5, 5], 1, use_torch=use_torch)
        nnet.train(X, T, n_epochs, method='adam', learning_rate=0.1)
        Y = nnet.use(X)
        print(f'adam {nnet.n_hidden_list} use_torch={use_torch} RMSE {rmse(Y, T):.3f} took {nnet.training_time:.3f} seconds')

plt.figure(1)
plt.clf()

nnet.draw()

```

Neural_Network.ipynb

The Neural_Network.ipynb Jupyter Notebook script uses the previously mentioned optimizers.py and neuralnetworks.py scripts. These supplementary scripts must be imported into the Neural_Network.ipynb script to have it operate correctly.

```
#Filename: Neural_Network.ipynb
#Authors: Noah Vento & Charles Anderson
#Description: Classify data using Neural Network algorithm and scripts

#Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import IPython.display as ipd # for display and clear_output
import time # for sleep

#Import Excel spreadsheet of data
Laguna_Fig = pd.read_excel('/Users/noahvento/Desktop/CSS_Noah/Database/M
L_Fig_Stats_Table.xlsx')

#Partition data into desired features
data = Laguna_Fig[['NTG', 'F4_Prop', 'F3_Prop', 'F2_Prop', 'F1_Prop',
                  'Gross', 'Drape', 'AR', 'Net', 'Num_Beds', 'P10_Phi_lcm',
                  'P50_Phi_lcm', 'P90_Phi_lcm', 'BT_min',
                  'BT_med', 'BT_max']]

#Make X have 16 columns and as many rows as needed to hold the values of
data
#Samples are in rows, and the features of each sample are in the columns
#The ith row of X is Sample i whose correct target output is in row i of
T
X = np.array(data).reshape((-1,16))

#Specify desired classification scheme and outputs
#(e.g., three-position classification scheme, where Axis = 1; Off-Axis =
2; Margin = 3)
Pos_3 = Laguna_Fig[['Pos_3']]

#Make T have one column and as many rows as needed to hold the values of
Pos_3
T = np.array(Pos_3).reshape((-1,1))

#To allow flexibility, decouple the modeling of the boundaries.
#Problem is due to using one value to represent all classes.
#Instead, use three values, one for each class.
#Binary-valued variables are adequate.
#Class 1 = (1,0,0), Class 2 = (0,1,0) and Class 3 = (0,0,1).
#These are called indicator variables.

def makeIndicatorVars(T):
    # Make sure T is two-dimensional. Should be nSamples x 1.
    if T.ndim == 1:
        T = T.reshape((-1,1))
    return (T == np.unique(T)).astype(int)
```

```

#Separate T into Axis, Off-Axis, and Margin data
axisI, _ = np.where(T == 1) # identifying axis
offaxisI, _ = np.where(T == 2) # identifying off-axis
marginI, _ = np.where(T == 3) # identifying margin

#Randomly permute Axis, Off-Axis, and Margin data
axisI = np.random.permutation(axisI)
offaxisI = np.random.permutation(offaxisI)
marginI = np.random.permutation(marginI)

#Generate fold (K = 5) for cross-validation
length_axisI = int(len(axisI)/5) #length of each fold
axis_folds = []
for i in range(4):
    axis_folds += [axisI[i*length_axisI:(i+1)*length_axisI]]
axis_folds += [axisI[4*length_axisI:len(axisI)]]

length_offaxisI = int(len(offaxisI)/5) #length of each fold
offaxis_folds = []
for i in range(4):
    offaxis_folds += [offaxisI[i*length_offaxisI:(i+1)*length_offaxisI]]
offaxis_folds += [offaxisI[4*length_offaxisI:len(offaxisI)]]

length_margin = int(len(marginI)/5) #length of each fold
margin_folds = []
for i in range(4):
    margin_folds += [marginI[i*length_margin:(i+1)*length_margin]]
margin_folds += [marginI[4*length_margin:len(marginI)]]

#Convert lists to arrays
axis_folds = np.array(axis_folds)
offaxis_folds = np.array(offaxis_folds)
margin_folds = np.array(margin_folds)

#Partition Axis Folds
axisfold_1 = axis_folds[0]
axisfold_2 = axis_folds[1]
axisfold_3 = axis_folds[2]
axisfold_4 = axis_folds[3]
axisfold_5 = axis_folds[4]

#Partition Off-Axis Folds
offaxisfold_1 = offaxis_folds[0]
offaxisfold_2 = offaxis_folds[1]
offaxisfold_3 = offaxis_folds[2]
offaxisfold_4 = offaxis_folds[3]
offaxisfold_5 = offaxis_folds[4]

#Partition Margin Folds
marginfold_1 = margin_folds[0]
marginfold_2 = margin_folds[1]
marginfold_3 = margin_folds[2]
marginfold_4 = margin_folds[3]
marginfold_5 = margin_folds[4]

```

```

#Set up training and testing folds
rowsTrain_1 = np.hstack((axisfold_1, axisfold_2, axisfold_3, axisfold_4,
                        offaxisfold_1, offaxisfold_2, offaxisfold_3,
                        offaxisfold_4, marginfold_1, marginfold_2,
                        marginfold_3, marginfold_4))
rowsTest_1 = np.hstack((axisfold_5, offaxisfold_5, marginfold_5))

rowsTrain_2 = np.hstack((axisfold_1, axisfold_2, axisfold_3, axisfold_5,
                        offaxisfold_1, offaxisfold_2, offaxisfold_3,
                        offaxisfold_5, marginfold_1, marginfold_2,
                        marginfold_3, marginfold_5))
rowsTest_2 = np.hstack((axisfold_4, offaxisfold_4, marginfold_4))

rowsTrain_3 = np.hstack((axisfold_1, axisfold_2, axisfold_4, axisfold_5,
                        offaxisfold_1, offaxisfold_2, offaxisfold_4,
                        offaxisfold_5, marginfold_1, marginfold_2,
                        marginfold_4, marginfold_5))
rowsTest_3 = np.hstack((axisfold_3, offaxisfold_3, marginfold_3))

rowsTrain_4 = np.hstack((axisfold_1, axisfold_3, axisfold_4, axisfold_5,
                        offaxisfold_1, offaxisfold_3, offaxisfold_4,
                        offaxisfold_5, marginfold_1, marginfold_3,
                        marginfold_4, marginfold_5))
rowsTest_4 = np.hstack((axisfold_2, offaxisfold_2, marginfold_2))

rowsTrain_5 = np.hstack((axisfold_2, axisfold_3, axisfold_4, axisfold_5,
                        offaxisfold_2, offaxisfold_3, offaxisfold_4,
                        offaxisfold_5, marginfold_2, marginfold_3,
                        marginfold_4, marginfold_5))
rowsTest_5 = np.hstack((axisfold_1, offaxisfold_1, marginfold_1))

```

```

#Xtrain = Array of training features from each channel element randomly
#          permuted in rowsTrain
#Ttrain = Corresponding outputs (classifications) for Xtrain values
Xtrain_1 = X[rowsTrain_1,:]
Ttrain_1 = T[rowsTrain_1,:]
Xtest_1 = X[rowsTest_1,:]
Ttest_1 = T[rowsTest_1,:]

Xtrain_2 = X[rowsTrain_2,:]
Ttrain_2 = T[rowsTrain_2,:]
Xtest_2 = X[rowsTest_2,:]
Ttest_2 = T[rowsTest_2,:]

Xtrain_3 = X[rowsTrain_3,:]
Ttrain_3 = T[rowsTrain_3,:]
Xtest_3 = X[rowsTest_3,:]
Ttest_3 = T[rowsTest_3,:]

Xtrain_4 = X[rowsTrain_4,:]
Ttrain_4 = T[rowsTrain_4,:]
Xtest_4 = X[rowsTest_4,:]
Ttest_4 = T[rowsTest_4,:]

Xtrain_5 = X[rowsTrain_5,:]
Ttrain_5 = T[rowsTrain_5,:]
Xtest_5 = X[rowsTest_5,:]
Ttest_5 = T[rowsTest_5,:]

```

```

#Define standardize function and standardize input data before classification
def standardize(X,mean,std):
    return (X - mean)/std

meanstrain_1, stdstrain_1 = np.mean(Xtrain_1, 0), np.std(Xtrain_1 ,0)
meanstest_1, stdtest_1 = np.mean(Xtest_1, 0), np.std(Xtest_1 ,0)
Xtrains_1 = standardize(Xtrain_1, meanstrain_1, stdstrain_1)
Xtests_1 = standardize(Xtest_1, meanstest_1, stdtest_1)

meanstrain_2, stdstrain_2 = np.mean(Xtrain_2, 0), np.std(Xtrain_2 ,0)
meanstest_2, stdtest_2 = np.mean(Xtest_2, 0), np.std(Xtest_2 ,0)
Xtrains_2 = standardize(Xtrain_2, meanstrain_2, stdstrain_2)
Xtests_2 = standardize(Xtest_2, meanstest_2, stdtest_2)

meanstrain_3, stdstrain_3 = np.mean(Xtrain_3, 0), np.std(Xtrain_3 ,0)
meanstest_3, stdtest_3 = np.mean(Xtest_3, 0), np.std(Xtest_3 ,0)
Xtrains_3 = standardize(Xtrain_3, meanstrain_3, stdstrain_3)
Xtests_3 = standardize(Xtest_3, meanstest_3, stdtest_3)

meanstrain_4, stdstrain_4 = np.mean(Xtrain_4, 0), np.std(Xtrain_4 ,0)
meanstest_4, stdtest_4 = np.mean(Xtest_4, 0), np.std(Xtest_4 ,0)
Xtrains_4 = standardize(Xtrain_4, meanstrain_4, stdstrain_4)
Xtests_4 = standardize(Xtest_4, meanstest_4, stdtest_4)

meanstrain_5, stdstrain_5 = np.mean(Xtrain_5, 0), np.std(Xtrain_5 ,0)
meanstest_5, stdtest_5 = np.mean(Xtest_5, 0), np.std(Xtest_5 ,0)
Xtrains_5 = standardize(Xtrain_5, meanstrain_5, stdstrain_5)
Xtests_5 = standardize(Xtest_5, meanstest_5, stdtest_5)

#Import Neural Network from neuralnetworks.py script
import neuralnetworks as nn
import imp
imp.reload(nn) # in case neuralnetworks.py has been changed

#One hidden layer with 5 neurons
n_hidden_list = [5]

#16 features in the input layer, n_hidden_list in the hidden layer,
#and three output classes in the output layer
nnet_1 = nn.NeuralNetworkClassifier(16, n_hidden_list, classes=[1, 2, 3
])

#train for 10 epochs using the first training fold
nnet_1.train(Xtrains_1, Ttrain_1 ,n_epochs=10, verbose=True)

#Train second Neural Network on training fold #2
nnet_2 = nn.NeuralNetworkClassifier(16, n_hidden_list, classes=[1, 2, 3
])
nnet_2.train(Xtrains_2, Ttrain_2 ,n_epochs=10, verbose=True)

```



```

#Train third Neural Network on training fold #3
nnet_3 = nn.NeuralNetworkClassifier(16, n_hiddens_list, classes=[1, 2, 3
])
nnet_3.train(Xtrains_3, Ttrain_3 ,n_epochs=10, verbose=True)

#Train fourth Neural Network on training fold #4
nnet_4 = nn.NeuralNetworkClassifier(16, n_hiddens_list, classes=[1, 2, 3
])
nnet_4.train(Xtrains_4, Ttrain_4 ,n_epochs=10, verbose=True)

#Train fifth Neural Network on training fold #5
nnet_5 = nn.NeuralNetworkClassifier(16, n_hiddens_list, classes=[1, 2, 3
])
nnet_5.train(Xtrains_5, Ttrain_5 ,n_epochs=10, verbose=True)

#Classify testing data for each fold
#Ytest = output classifications
#Ptest = output probabilities
Ytest_1, Ptest_1 = nnet_1.use(Xtests_1)
Ytest_2, Ptest_2 = nnet_2.use(Xtests_2)
Ytest_3, Ptest_3 = nnet_3.use(Xtests_3)
Ytest_4, Ptest_4 = nnet_4.use(Xtests_4)
Ytest_5, Ptest_5 = nnet_5.use(Xtests_5)

#Calculate accuracies of each test fold by comparing to true labels
testacc_1 = (np.sum(Ytest_1 == Ttest_1)/len(Ttest_1))*100
testacc_2 = (np.sum(Ytest_2 == Ttest_2)/len(Ttest_2))*100
testacc_3 = (np.sum(Ytest_3 == Ttest_3)/len(Ttest_3))*100
testacc_4 = (np.sum(Ytest_4 == Ttest_4)/len(Ttest_4))*100
testacc_5 = (np.sum(Ytest_5 == Ttest_5)/len(Ttest_5))*100

#Find highest accuracy from test folds
highest_acc = max((testacc_1, testacc_2, testacc_3, testacc_4, testacc_5
))

#Concatenate all folds to calculate accuracy
Ytest_tot = np.concatenate((Ytest_1, Ytest_2, Ytest_3, Ytest_4, Ttest_5
))
Ttest_tot = np.concatenate((Ttest_1, Ttest_2, Ttest_3, Ttest_4, Ttest_5
))
Ptest_tot = np.concatenate((Ptest_1, Ptest_2, Ptest_3, Ptest_4, Ptest_5
))

#Calculate accuracy
crossval_acc = (np.sum(Ytest_tot == Ttest_tot)/len(Ttest_tot))*100

#Print cross-validation accuracy and highest accuracy
crossval_acc, highest_acc

```

APPENDIX C: MACHINE LEARNING RESULTS

Description

Various machine learning algorithms—unsupervised k-means, 26 variations of common supervised learners, and a deep learning neural network—were used to classify the deep-water channel outcrop data from the Laguna Figueroa database in this study. All algorithms except for k-means, XGBoost, random forest, and the neural network were operated in MATLAB's Classification Learner app, which is a part of the Statistics and Machine Learning Toolbox. The Classification Learner app trains models to classify data using supervised machine learning algorithms and makes it easy to import data tables, partition data into training and testing, and run a suite of algorithms quickly.

The remaining algorithms were operated in Python using the Scikit-learn machine learning library, which features a wide range of classification, regression, and clustering algorithms. The Python codes for these algorithms can be viewed in Appendix B. The results and evaluation metrics for all of the machine learning analyses can be viewed in this section.

Table C.1 FD-2P for unsupervised and supervised learning algorithms.

Algorithm	Parameters	Accuracy
K-Means	Number of Clusters = 2	78.98%
Fine Tree	Maximum Number of Splits: 100	85.40%
Medium Tree	Maximum Number of Splits: 20	85.40%
Coarse Tree	Maximum Number of Splits: 4	85.40%
Linear Discriminant Quadratic Discriminant	Covariance Structure: Full	89.20%
Gaussian Naïve Bayes	-	88.50%
Kernel Naïve Bayes	-	88.50%
Linear SVM	Box Constraint = 1	91.70%
Quadratic SVM	Box Constraint = 1	90.40%
Cubic SVM	Box Constraint = 1	89.20%
Fine Gaussian SVM	Box Constraint = 1; Kernel Scale = 1.0	71.30%
Medium Gaussian SVM	Box Constraint = 1; Kernel Scale = 4.0	89.80%
Coarse Gaussian SVM	Box Constraint = 3; Kernel Scale = 16	84.70%
Fine KNN	Nearest Neighbors = 1	85.40%
Medium KNN	Nearest Neighbors = 10	87.90%
Coarse KNN	Nearest Neighbors = 50	82.20%
Cosine KNN	Nearest Neighbors = 10	89.80%
Cubic KNN	Nearest Neighbors = 10	87.90%
Weighted KNN	Nearest Neighbors = 10	90.40%
AdaBoost Trees	Max. Number of Splits: 20; Number of Learners: 30	72.00%
Bagged Trees	Max. Number of Splits: 156; Number of Learners: 30	85.40%
Subspace Discriminant	Number of Learners: 30; Subspace Dimensions: 8	87.90%
Subspace KNN	Number of Learners: 30; Subspace Dimensions: 8	89.20%
RUSBoosted Trees	Max. Number of Splits: 20; Number of Learners: 30	91.10%
Random Forest	-	91.72%
XGBoost	-	90.45%
Neural Network	Number of Neurons in Hidden Layer: 5	93.63%

Table C.2 FD-2P precision, recall, and F1 score metrics for unsupervised and supervised learning algorithms.

Algorithm	Class	Precision	Recall	F1 Score
K-Means*	Axis	0.84	0.87	0.86
	Margin	0.63	0.59	0.61
Fine Tree†	Axis	0.91	0.88	0.90
	Margin	0.72	0.77	0.75
Medium Tree†	Axis	0.91	0.88	0.90
	Margin	0.72	0.77	0.75
Coarse Tree†	Axis	0.90	0.89	0.90
	Margin	0.73	0.75	0.74
Linear Discriminant†	Axis	0.93	0.92	0.92
	Margin	0.80	0.82	0.81
Quadratic Discriminant†	Axis	0.97	0.87	0.92
	Margin	0.73	0.93	0.82
Gaussian Naïve Bayes†	Axis	0.97	0.87	0.92
	Margin	0.73	0.93	0.82
Kernel Naïve Bayes†	Axis	0.94	0.90	0.92
	Margin	0.77	0.84	0.80
Linear SVM†	Axis	0.93	0.96	0.94
	Margin	0.88	0.82	0.85
Quadratic SVM†	Axis	0.93	0.94	0.93
	Margin	0.84	0.82	0.83
Cubic SVM†	Axis	0.93	0.92	0.92
	Margin	0.80	0.82	0.81
Fine Gaussian SVM†	Axis	0.72	0.99	0.83
	Margin	0.00	0.00	-
Medium Gaussian SVM†	Axis	0.94	0.92	0.93
	Margin	0.80	0.84	0.82
Coarse Gaussian SVM†	Axis	0.84	0.97	0.90
	Margin	0.88	0.52	0.66
Fine KNN†	Axis	0.89	0.90	0.90
	Margin	0.74	0.73	0.74
Medium KNN†	Axis	0.89	0.95	0.92
	Margin	0.84	0.70	0.77
Coarse KNN†	Axis	0.81	0.99	0.89
	Margin	0.94	0.39	0.55
Cosine KNN†	Axis	0.94	0.92	0.93
	Margin	0.80	0.84	0.82

Algorithm	Class	Precision	Recall	F1 Score
Cubic KNN†	Axis	0.89	0.95	0.92
	Margin	0.84	0.70	0.77
Weighted KNN†	Axis	0.94	0.93	0.93
	Margin	0.82	0.84	0.83
AdaBoost Trees†§	Axis	0.72	1.00	0.84
	Margin	-	0.00	-
Bagged Trees†§	Axis	0.89	0.91	0.90
	Margin	0.76	0.70	0.73
Subspace Discriminant†§	Axis	0.89	0.95	0.92
	Margin	0.84	0.70	0.77
Subspace KNN†§	Axis	0.91	0.94	0.93
	Margin	0.83	0.77	0.80
RUSBoosted Trees†§	Axis	0.96	0.91	0.94
	Margin	0.80	0.91	0.85
Random Forest†§	Axis	0.94	0.95	0.94
	Margin	0.86	0.84	0.85
XGBoost†§	Axis	0.94	0.93	0.93
	Margin	0.82	0.84	0.83
Neural Network†‡	Axis	0.96	0.96	0.96
	Margin	0.89	0.89	0.89

*Unsupervised Learning; †Supervised Learning; §Ensemble Classifier; ‡Deep Learning

Table C.3 FD-3P accuracies for unsupervised and supervised learning algorithms.

Algorithm	Parameters	Accuracy
K-Means*	Number of Clusters = 3	48.41%
Fine Tree†	Maximum Number of Splits: 100	63.10%
Medium Tree†	Maximum Number of Splits: 20	63.10%
Coarse Tree†	Maximum Number of Splits: 4	65.00%
Linear Discriminant†	Covariance Structure: Full	78.30%
Quadratic Discriminant†	Covariance Structure: Diagonal	81.50%
Gaussian Naïve Bayes†	-	81.50%
Kernel Naïve Bayes†	-	59.90%
Linear SVM†	Box Constraint = 1	82.80%
Quadratic SVM†	Box Constraint = 1	78.30%
Cubic SVM†	Box Constraint = 1	74.50%
Fine Gaussian SVM†	Box Constraint = 1; Kernel Scale = 1.0	44.60%
Medium Gaussian SVM†	Box Constraint = 1; Kernel Scale = 4.0	79.60%
Coarse Gaussian SVM†	Box Constraint = 3; Kernel Scale = 16	73.20%
Fine KNN†	Nearest Neighbors = 1	68.80%
Medium KNN†	Nearest Neighbors = 10	77.70%
Coarse KNN†	Nearest Neighbors = 50	73.20%
Cosine KNN†	Nearest Neighbors = 10	75.80%
Cubic KNN†	Nearest Neighbors = 10	78.30%
Weighted KNN†	Nearest Neighbors = 10	77.70%
AdaBoost Trees†§	Max. Number of Splits: 20; Number of Learners: 30	65.60%
Bagged Trees†§	Max. Number of Splits: 156; Number of Learners: 30	75.20%
Subspace Discriminant†§	Number of Learners: 30; Subspace Dimensions: 8	79.60%
Subspace KNN†§	Number of Learners: 30; Subspace Dimensions: 8	76.40%
RUSBoosted Trees†§	Max. Number of Splits: 20; Number of Learners: 30	73.90%
Random Forest†§	-	82.80%
XGBoost†§	-	80.25%
Neural Network†‡	Number of Neurons in Hidden Layer: 5	80.25%

*Unsupervised Learning; †Supervised Learning; §Ensemble Classifier; ‡Deep Learning

Table C.4 FD-3P precision, recall, and F1 score metrics for unsupervised and supervised learning algorithms.

Algorithm	Class	Precision	Recall	F1 Score
K-Means*	Axis	0.49	0.75	0.59
	Off-Axis	0.43	0.44	0.43
	Margin	0.65	0.25	0.36
Fine Tree†	Axis	0.70	0.73	0.71
	Off-Axis	0.57	0.48	0.52
	Margin	0.63	0.73	0.67
Medium Tree†	Axis	0.70	0.73	0.71
	Off-Axis	0.57	0.48	0.52
	Margin	0.63	0.73	0.67
Coarse Tree†	Axis	0.72	0.65	0.68
	Off-Axis	0.56	0.55	0.55
	Margin	0.70	0.80	0.74
Linear Discriminant†	Axis	0.87	0.76	0.81
	Off-Axis	0.71	0.79	0.75
	Margin	0.81	0.80	0.80
Quadratic Discriminant†	Axis	0.83	0.84	0.83
	Off-Axis	0.75	0.81	0.78
	Margin	0.92	0.80	0.85
Gaussian Naïve Bayes†	Axis	0.83	0.84	0.83
	Off-Axis	0.75	0.81	0.78
	Margin	0.92	0.80	0.85
Kernel Naïve Bayes†	Axis	0.55	0.53	0.54
	Off-Axis	0.52	0.58	0.55
	Margin	0.79	0.70	0.75
Linear SVM†	Axis	0.85	0.80	0.83
	Off-Axis	0.78	0.79	0.78
	Margin	0.87	0.91	0.89
Quadratic SVM†	Axis	0.85	0.78	0.82
	Off-Axis	0.71	0.76	0.73
	Margin	0.82	0.82	0.82
Cubic SVM†	Axis	0.80	0.73	0.76
	Off-Axis	0.67	0.69	0.68
	Margin	0.79	0.84	0.81
Fine Gaussian SVM†	Axis	0.79	0.22	0.34
	Off-Axis	0.41	0.92	0.57
	Margin	0.50	0.05	0.08

Algorithm	Class	Precision	Recall	F1 Score
Medium Gaussian SVM†	Axis	0.83	0.78	0.81
	Off-Axis	0.74	0.74	0.74
	Margin	0.83	0.89	0.86
Coarse Gaussian SVM†	Axis	0.92	0.67	0.77
	Off-Axis	0.61	0.89	0.72
	Margin	0.87	0.59	0.70
Fine KNN†	Axis	0.77	0.67	0.72
	Off-Axis	0.60	0.65	0.62
	Margin	0.74	0.77	0.76
Medium KNN†	Axis	0.83	0.78	0.81
	Off-Axis	0.70	0.77	0.73
	Margin	0.85	0.77	0.81
Coarse KNN†	Axis	0.91	0.63	0.74
	Off-Axis	0.60	0.94	0.73
	Margin	0.96	0.57	0.71
Cosine KNN†	Axis	0.78	0.84	0.81
	Off-Axis	0.72	0.63	0.67
	Margin	0.77	0.84	0.80
Cubic KNN†	Axis	0.85	0.76	0.80
	Off-Axis	0.70	0.79	0.74
	Margin	0.85	0.80	0.82
Weighted KNN†	Axis	0.84	0.73	0.78
	Off-Axis	0.70	0.77	0.73
	Margin	0.84	0.84	0.84
AdaBoost Trees†§	Axis	0.71	0.76	0.74
	Off-Axis	0.59	0.53	0.56
	Margin	0.67	0.70	0.69
Bagged Trees†§	Axis	0.83	0.78	0.81
	Off-Axis	0.67	0.76	0.71
	Margin	0.79	0.70	0.75
Subspace Discriminant†§	Axis	0.85	0.80	0.83
	Off-Axis	0.71	0.81	0.76
	Margin	0.87	0.77	0.82
Subspace KNN†§	Axis	0.82	0.71	0.76
	Off-Axis	0.69	0.74	0.71
	Margin	0.83	0.86	0.84
RUSBoosted Trees†§	Axis	0.80	0.84	0.82
	Off-Axis	0.69	0.61	0.65
	Margin	0.73	0.80	0.76

Algorithm	Class	Precision	Recall	F1 Score
Random Forest†§	Axis	0.87	0.80	0.84
	Off-Axis	0.78	0.79	0.78
	Margin	0.85	0.91	0.88
XGBoost †§	Axis	0.86	0.82	0.84
	Off-Axis	0.74	0.79	0.77
	Margin	0.83	0.80	0.81
Neural Network†‡	Axis	0.81	0.84	0.83
	Off-Axis	0.79	0.71	0.75
	Margin	0.81	0.89	0.85

*Unsupervised Learning; †Supervised Learning; §Ensemble Classifier; ‡Deep Learning

Table C.5 GM-2P-1 classification accuracies for unsupervised and supervised learning algorithms.

Algorithm	Parameters	Accuracy
K-Means*	Number of Clusters = 2	71.34%
Fine Tree†	Maximum Number of Splits: 100	74.50%
Medium Tree†	Maximum Number of Splits: 20	74.50%
Coarse Tree†	Maximum Number of Splits: 4	80.30%
Linear Discriminant†	Covariance Structure: Full	81.50%
Quadratic Discriminant†	Covariance Structure: Diagonal	75.20%
Gaussian Naïve Bayes†	-	75.20%
Kernel Naïve Bayes†	-	75.80%
Linear SVM†	Box Constraint = 1	84.10%
Quadratic SVM†	Box Constraint = 1	81.50%
Cubic SVM†	Box Constraint = 1	80.90%
Fine Gaussian SVM†	Box Constraint = 1; Kernel Scale = 1.0	83.40%
Medium Gaussian SVM†	Box Constraint = 1; Kernel Scale = 4.0	84.10%
Coarse Gaussian SVM†	Box Constraint = 3; Kernel Scale = 16	83.40%
Fine KNN†	Nearest Neighbors = 1	84.70%
Medium KNN†	Nearest Neighbors = 10	83.40%
Coarse KNN†	Nearest Neighbors = 50	83.40%
Cosine KNN†	Nearest Neighbors = 10	83.40%
Cubic KNN†	Nearest Neighbors = 10	83.40%
Weighted KNN†	Nearest Neighbors = 10	84.10%
AdaBoost Trees†§	Max. Number of Splits: 20; Number of Learners: 30	83.40%
Bagged Trees†§	Max. Number of Splits: 156; Number of Learners: 30	84.70%
Subspace Discriminant†§	Number of Learners: 30; Subspace Dimensions: 8	83.40%
Subspace KNN†§	Number of Learners: 30; Subspace Dimensions: 8	84.70%
RUSBoosted Trees†§	Max. Number of Splits: 20; Number of Learners: 30	65.60%
Random Forest†§	-	85.99%
XGBoost†§	-	85.99%
Neural Network†‡	Number of Neurons in Hidden Layer: 5	85.99%

*Unsupervised Learning; †Supervised Learning; §Ensemble Classifier; ‡Deep Learning

Table C.6 GM-2P-1 precision, recall, and F1 score metrics for unsupervised and supervised learning algorithms.

Algorithm	Class	Precision	Recall	F1 Score
K-Means*	Axis	0.87	0.77	0.82
	Margin	0.27	0.42	0.33
Fine Tree†	Axis	0.84	0.85	0.85
	Margin	0.21	0.19	0.20
Medium Tree†	Axis	0.84	0.85	0.85
	Margin	0.21	0.19	0.20
Coarse Tree†	Axis	0.85	0.93	0.89
	Margin	0.31	0.15	0.21
Linear Discriminant†	Axis	0.84	0.95	0.90
	Margin	0.33	0.12	0.17
Quadratic Discriminant†	Axis	0.89	0.80	0.84
	Margin	0.33	0.50	0.40
Gaussian Naïve Bayes†	Axis	0.89	0.80	0.84
	Margin	0.33	0.50	0.40
Kernel Naïve Bayes†	Axis	0.87	0.83	0.85
	Margin	0.31	0.38	0.34
Linear SVM†	Axis	0.85	0.98	0.91
	Margin	0.60	0.12	0.19
Quadratic SVM†	Axis	0.87	0.92	0.89
	Margin	0.42	0.31	0.36
Cubic SVM†	Axis	0.87	0.91	0.89
	Margin	0.40	0.31	0.35
Fine Gaussian SVM†	Axis	0.83	1.00	0.91
	Margin	-	0.00	-
Medium Gaussian SVM†	Axis	0.84	1.00	0.91
	Margin	1.00	0.04	0.07
Coarse Gaussian SVM†	Axis	0.83	1.00	0.91
	Margin	-	0.00	-
Fine KNN†	Axis	0.88	0.94	0.91
	Margin	0.56	0.38	0.45
Medium KNN†	Axis	0.83	1.00	0.91
	Margin	-	0.00	-
Coarse KNN†	Axis	0.83	1.00	0.91
	Margin	-	0.00	-
Cosine KNN†	Axis	0.83	1.00	0.91
	Margin	-	0.00	-

Algorithm	Class	Precision	Recall	F1 Score
Cubic KNN†	Axis	0.83	1.00	0.91
	Margin	-	0.00	-
Weighted KNN†	Axis	0.84	0.99	0.91
	Margin	0.67	0.08	0.14
AdaBoost Trees†§	Axis	0.83	1.00	0.91
	Margin	-	0.00	-
Bagged Trees†§	Axis	0.86	0.98	0.91
	Margin	0.63	0.19	0.29
Subspace Discriminant†§	Axis	0.85	0.97	0.91
	Margin	0.50	0.15	0.24
Subspace KNN†§	Axis	0.87	0.96	0.91
	Margin	0.58	0.27	0.37
RUSBoosted Trees†§	Axis	0.85	0.71	0.78
	Margin	0.21	0.38	0.27
Random Forest†§	Axis	0.87	0.98	0.92
	Margin	0.70	0.27	0.39
XGBoost†§	Axis	0.88	0.96	0.92
	Margin	0.64	0.35	0.45
Neural Network†‡	Axis	0.88	0.96	0.92
	Margin	0.64	0.35	0.45

*Unsupervised Learning; †Supervised Learning; §Ensemble Classifier; ‡Deep Learning

Table C.7 GM-2P-2 classification accuracies for unsupervised and supervised learning algorithms.

Algorithm	Parameters	Accuracy
K-Means*	Number of Clusters = 2	63.06%
Fine Tree†	Maximum Number of Splits: 100	63.70%
Medium Tree†	Maximum Number of Splits: 20	63.70%
Coarse Tree†	Maximum Number of Splits: 4	69.40%
Linear Discriminant†	Covariance Structure: Full	65.00%
Quadratic Discriminant†	Covariance Structure: Diagonal	66.90%
Gaussian Naïve Bayes†	-	66.90%
Kernel Naïve Bayes†	-	59.90%
Linear SVM†	Box Constraint = 1	70.10%
Quadratic SVM†	Box Constraint = 1	59.20%
Cubic SVM†	Box Constraint = 1	59.90%
Fine Gaussian SVM†	Box Constraint = 1; Kernel Scale = 1.0	69.40%
Medium Gaussian SVM†	Box Constraint = 1; Kernel Scale = 4.0	68.80%
Coarse Gaussian SVM†	Box Constraint = 3; Kernel Scale = 16	70.10%
Fine KNN†	Nearest Neighbors = 1	56.10%
Medium KNN†	Nearest Neighbors = 10	67.50%
Coarse KNN†	Nearest Neighbors = 50	70.10%
Cosine KNN†	Nearest Neighbors = 10	67.50%
Cubic KNN†	Nearest Neighbors = 10	68.20%
Weighted KNN†	Nearest Neighbors = 10	60.50%
AdaBoost Trees†§	Max. Number of Splits: 20; Number of Learners: 30	64.30%
Bagged Trees†§	Max. Number of Splits: 156; Number of Learners: 30	68.20%
Subspace Discriminant†§	Number of Learners: 30; Subspace Dimensions: 8	69.40%
Subspace KNN†§	Number of Learners: 30; Subspace Dimensions: 8	68.20%
RUSBoosted Trees†§	Max. Number of Splits: 20; Number of Learners: 30	49.00%
Random Forest†§	-	75.16%
XGBoost†§	-	73.25%
Neural Network†‡	Number of Neurons in Hidden Layer: 5	75.16%

*Unsupervised Learning; †Supervised Learning; §Ensemble Classifier; ‡Deep Learning

Table C.8 GM-2P-2 precision, recall, and F1 score metrics for unsupervised and supervised learning algorithms.

Algorithm	Class	Precision	Recall	F1 Score
K-Means*	Axis	0.72	0.76	0.74
	Margin	0.37	0.32	0.34
Fine Tree†	Axis	0.75	0.73	0.74
	Margin	0.40	0.43	0.41
Medium Tree†	Axis	0.75	0.73	0.74
	Margin	0.40	0.43	0.41
Coarse Tree†	Axis	0.72	0.92	0.81
	Margin	0.47	0.17	0.25
Linear Discriminant†	Axis	0.70	0.88	0.78
	Margin	0.28	0.11	0.15
Quadratic Discriminant†	Axis	0.73	0.84	0.78
	Margin	0.42	0.28	0.33
Gaussian Naïve Bayes†	Axis	0.73	0.84	0.78
	Margin	0.42	0.28	0.33
Kernel Naïve Bayes†	Axis	0.69	0.76	0.73
	Margin	0.28	0.21	0.24
Linear SVM†	Axis	0.71	0.98	0.82
	Margin	0.50	0.04	0.08
Quadratic SVM†	Axis	0.69	0.75	0.72
	Margin	0.28	0.23	0.26
Cubic SVM†	Axis	0.71	0.72	0.71
	Margin	0.33	0.32	0.32
Fine Gaussian SVM†	Axis	0.70	0.99	0.82
	Margin	0.00	0.00	-
Medium Gaussian SVM†	Axis	0.70	0.97	0.81
	Margin	0.25	0.02	0.04
Coarse Gaussian SVM†	Axis	0.70	1.00	0.82
	Margin	-	0.00	-
Fine KNN†	Axis	0.68	0.71	0.69
	Margin	0.24	0.21	0.22
Medium KNN†	Axis	0.70	0.95	0.80
	Margin	0.25	0.04	0.07
Coarse KNN†	Axis	0.70	1.00	0.82
	Margin	-	0.00	-
Cosine KNN†	Axis	0.70	0.94	0.80
	Margin	0.30	0.06	0.11

Algorithm	Class	Precision	Recall	F1 Score
Cubic KNN†	Axis	0.69	0.97	0.81
	Margin	0.00	0.00	-
Weighted KNN†	Axis	0.68	0.84	0.75
	Margin	0.14	0.06	0.09
AdaBoost Trees†§	Axis	0.71	0.82	0.76
	Margin	0.35	0.23	0.28
Bagged Trees†§	Axis	0.72	0.89	0.80
	Margin	0.43	0.19	0.26
Subspace Discriminant†§	Axis	0.71	0.96	0.82
	Margin	0.43	0.06	0.11
Subspace KNN†§	Axis	0.75	0.83	0.78
	Margin	0.46	0.34	0.39
RUSBoosted Trees†§	Axis	0.69	0.49	0.57
	Margin	0.29	0.49	0.37
Random Forest†§	Axis	0.77	0.92	0.84
	Margin	0.65	0.36	0.47
XGBoost†§	Axis	0.77	0.89	0.82
	Margin	0.59	0.36	0.45
Neural Network†‡	Axis	0.77	0.93	0.84
	Margin	0.67	0.34	0.45

*Unsupervised Learning; †Supervised Learning; §Ensemble Classifier; ‡Deep Learning

Table C.9 GM-3P-1 classification accuracies for unsupervised and supervised learning algorithms.

Algorithm	Parameters	Accuracy
K-Means*	Number of Clusters = 3	55.41%
Fine Tree†	Maximum Number of Splits: 100	61.80%
Medium Tree†	Maximum Number of Splits: 20	61.80%
Coarse Tree†	Maximum Number of Splits: 4	66.90%
Linear Discriminant†	Covariance Structure: Full	66.90%
Quadratic Discriminant†	Covariance Structure: Diagonal	61.80%
Gaussian Naïve Bayes†	-	61.80%
Kernel Naïve Bayes†	-	56.10%
Linear SVM†	Box Constraint = 1	69.40%
Quadratic SVM†	Box Constraint = 1	62.40%
Cubic SVM†	Box Constraint = 1	59.90%
Fine Gaussian SVM†	Box Constraint = 1; Kernel Scale = 1.0	70.10%
Medium Gaussian SVM†	Box Constraint = 1; Kernel Scale = 4.0	69.40%
Coarse Gaussian SVM†	Box Constraint = 3; Kernel Scale = 16	70.10%
Fine KNN†	Nearest Neighbors = 1	58.00%
Medium KNN†	Nearest Neighbors = 10	68.80%
Coarse KNN†	Nearest Neighbors = 50	70.10%
Cosine KNN†	Nearest Neighbors = 10	65.60%
Cubic KNN†	Nearest Neighbors = 10	67.50%
Weighted KNN†	Nearest Neighbors = 10	64.30%
AdaBoost Trees†§	Max. Number of Splits: 20; Number of Learners: 30	61.80%
Bagged Trees†§	Max. Number of Splits: 156; Number of Learners: 30	65.00%
Subspace Discriminant†§	Number of Learners: 30; Subspace Dimensions: 8	70.10%
Subspace KNN†§	Number of Learners: 30; Subspace Dimensions: 8	68.80%
RUSBoosted Trees†§	Max. Number of Splits: 20; Number of Learners: 30	49.00%
Random Forest†§	-	71.97%
XGBoost†§	-	73.89%
Neural Network†‡	Number of Neurons in Hidden Layer: 5	73.89%

*Unsupervised Learning; †Supervised Learning; §Ensemble Classifier; ‡Deep Learning

Table C.10 GM-3P-1 precision, recall, and F1 score metrics for unsupervised and supervised learning algorithms.

Algorithm	Class	Precision	Recall	F1 Score
K-Means*	Axis	0.71	0.65	0.68
	Off-Axis	0.29	0.42	0.34
	Margin	0.00	0.00	-
Fine Tree†	Axis	0.72	0.76	0.74
	Off-Axis	0.30	0.28	0.29
	Margin	0.43	0.27	0.33
Medium Tree†	Axis	0.72	0.76	0.74
	Off-Axis	0.30	0.28	0.29
	Margin	0.43	0.27	0.33
Coarse Tree†	Axis	0.70	0.91	0.79
	Off-Axis	0.36	0.14	0.20
	Margin	0.00	0.00	-
Linear Discriminant†	Axis	0.71	0.89	0.79
	Off-Axis	0.29	0.11	0.16
	Margin	0.60	0.27	0.38
Quadratic Discriminant†	Axis	0.75	0.76	0.76
	Off-Axis	0.29	0.19	0.23
	Margin	0.29	0.55	0.38
Gaussian Naïve Bayes†	Axis	0.75	0.76	0.76
	Off-Axis	0.29	0.19	0.23
	Margin	0.29	0.55	0.38
Kernel Naïve Bayes†	Axis	0.71	0.70	0.70
	Off-Axis	0.21	0.22	0.21
	Margin	0.33	0.27	0.30
Linear SVM†	Axis	0.70	0.98	0.82
	Off-Axis	-	0.00	-
	Margin	0.33	0.09	0.14
Quadratic SVM†	Axis	0.71	0.81	0.75
	Off-Axis	0.25	0.17	0.20
	Margin	0.43	0.27	0.33
Cubic SVM†	Axis	0.72	0.74	0.73
	Off-Axis	0.29	0.28	0.28
	Margin	0.33	0.27	0.30
Fine Gaussian SVM†	Axis	0.70	1.00	0.82
	Off-Axis	-	0.00	-
	Margin	-	0.00	-

Algorithm	Class	Precision	Recall	F1 Score
Medium Gaussian SVM†	Axis	0.70	0.99	0.82
	Off-Axis	-	0.00	-
	Margin	0.00	0.00	-
Coarse Gaussian SVM†	Axis	0.70	1.00	0.82
	Off-Axis	-	0.00	-
	Margin	-	0.00	-
Fine KNN†	Axis	0.70	0.75	0.72
	Off-Axis	0.18	0.14	0.16
	Margin	0.33	0.36	0.35
Medium KNN†	Axis	0.70	0.96	0.81
	Off-Axis	0.33	0.06	0.10
	Margin	-	0.00	-
Coarse KNN†	Axis	0.70	1.00	0.82
	Off-Axis	-	0.00	-
	Margin	-	0.00	-
Cosine KNN†	Axis	0.69	0.91	0.79
	Off-Axis	0.25	0.08	0.13
	Margin	0.00	0.00	-
Cubic KNN†	Axis	0.70	0.95	0.81
	Off-Axis	0.22	0.06	0.09
	Margin	-	0.00	-
Weighted KNN†	Axis	0.71	0.85	0.77
	Off-Axis	0.26	0.14	0.18
	Margin	0.40	0.18	0.25
AdaBoost Trees†§	Axis	0.70	0.81	0.75
	Off-Axis	0.24	0.17	0.20
	Margin	0.40	0.18	0.25
Bagged Trees†§	Axis	0.70	0.88	0.78
	Off-Axis	0.20	0.08	0.12
	Margin	0.50	0.18	0.27
Subspace Discriminant†§	Axis	0.71	0.96	0.82
	Off-Axis	0.33	0.03	0.05
	Margin	0.60	0.27	0.38
Subspace KNN†§	Axis	0.75	0.87	0.81
	Off-Axis	0.38	0.25	0.30
	Margin	0.60	0.27	0.38
RUSBoosted Trees†§	Axis	0.75	0.47	0.58
	Off-Axis	0.30	0.47	0.37
	Margin	0.25	0.73	0.37

Algorithm	Class	Precision	Recall	F1 Score
Random Forest†§	Axis	0.75	0.92	0.82
	Off-Axis	0.53	0.25	0.34
	Margin	0.60	0.27	0.38
XGBoost †§	Axis	0.77	0.93	0.84
	Off-Axis	0.56	0.28	0.37
	Margin	0.67	0.36	0.47
Neural Network†‡	Axis	0.76	0.93	0.84
	Off-Axis	0.61	0.31	0.41
	Margin	0.60	0.27	0.38

*Unsupervised Learning; †Supervised Learning; §Ensemble Classifier; ‡Deep Learning

Table C.11 GM-3P-2 classification accuracies for unsupervised and supervised learning algorithms.

Algorithm	Parameters	Accuracy
K-Means*	Number of Clusters = 3	43.31%
Fine Tree†	Maximum Number of Splits: 100	44.60%
Medium Tree†	Maximum Number of Splits: 20	44.60%
Coarse Tree†	Maximum Number of Splits: 4	49.00%
Linear Discriminant†	Covariance Structure: Full	46.50%
Quadratic Discriminant†	Covariance Structure: Diagonal	47.80%
Gaussian Naïve Bayes†	-	47.80%
Kernel Naïve Bayes†	-	42.70%
Linear SVM†	Box Constraint = 1	47.80%
Quadratic SVM†	Box Constraint = 1	49.00%
Cubic SVM†	Box Constraint = 1	45.90%
Fine Gaussian SVM†	Box Constraint = 1; Kernel Scale = 1.0	51.00%
Medium Gaussian SVM†	Box Constraint = 1; Kernel Scale = 4.0	48.40%
Coarse Gaussian SVM†	Box Constraint = 3; Kernel Scale = 16	51.60%
Fine KNN†	Nearest Neighbors = 1	45.90%
Medium KNN†	Nearest Neighbors = 10	47.10%
Coarse KNN†	Nearest Neighbors = 50	51.60%
Cosine KNN†	Nearest Neighbors = 10	48.40%
Cubic KNN†	Nearest Neighbors = 10	47.80%
Weighted KNN†	Nearest Neighbors = 10	47.80%
AdaBoost Trees†§	Max. Number of Splits: 20; Number of Learners: 30	51.60%
Bagged Trees†§	Max. Number of Splits: 156; Number of Learners: 30	49.70%
Subspace Discriminant†§	Number of Learners: 30; Subspace Dimensions: 8	52.20%
Subspace KNN†§	Number of Learners: 30; Subspace Dimensions: 8	51.60%
RUSBoosted Trees†§	Max. Number of Splits: 20; Number of Learners: 30	41.40%
Random Forest†§	-	62.42%
XGBoost†§	-	62.42%
Neural Network†‡	Number of Neurons in Hidden Layer: 5	61.78%

*Unsupervised Learning; †Supervised Learning; §Ensemble Classifier; ‡Deep Learning

Table C.12 GM-3P-2 precision, recall, and F1 score metrics for unsupervised and supervised learning algorithms.

Algorithm	Class	Precision	Recall	F1 Score
K-Means*	Axis	0.52	0.59	0.55
	Off-Axis	0.31	0.41	0.36
	Margin	0.25	0.03	0.06
Fine Tree†	Axis	0.62	0.54	0.58
	Off-Axis	0.34	0.43	0.38
	Margin	0.21	0.20	0.21
Medium Tree†	Axis	0.62	0.54	0.58
	Off-Axis	0.34	0.43	0.38
	Margin	0.21	0.20	0.21
Coarse Tree†	Axis	0.54	0.79	0.64
	Off-Axis	0.40	0.22	0.28
	Margin	0.23	0.10	0.14
Linear Discriminant†	Axis	0.58	0.72	0.64
	Off-Axis	0.24	0.20	0.21
	Margin	0.32	0.20	0.24
Quadratic Discriminant†	Axis	0.59	0.58	0.58
	Off-Axis	0.29	0.26	0.28
	Margin	0.44	0.53	0.48
Gaussian Naïve Bayes†	Axis	0.59	0.58	0.58
	Off-Axis	0.29	0.26	0.28
	Margin	0.44	0.53	0.48
Kernel Naïve Bayes†	Axis	0.55	0.64	0.59
	Off-Axis	0.21	0.15	0.18
	Margin	0.27	0.27	0.27
Linear SVM†	Axis	0.51	0.83	0.63
	Off-Axis	0.18	0.07	0.10
	Margin	0.56	0.17	0.26
Quadratic SVM†	Axis	0.59	0.70	0.64
	Off-Axis	0.34	0.35	0.34
	Margin	0.29	0.13	0.18
Cubic SVM†	Axis	0.60	0.65	0.62
	Off-Axis	0.29	0.30	0.30
	Margin	0.25	0.17	0.20
Fine Gaussian SVM†	Axis	0.52	0.99	0.68
	Off-Axis	0.00	0.00	-
	Margin	-	0.00	-

Algorithm	Class	Precision	Recall	F1 Score
Medium Gaussian SVM†	Axis	0.52	0.85	0.65
	Off-Axis	0.26	0.11	0.15
	Margin	0.33	0.07	0.11
Coarse Gaussian SVM†	Axis	0.48	1.00	0.65
	Off-Axis	-	0.00	-
	Margin	-	0.00	-
Fine KNN†	Axis	0.62	0.59	0.61
	Off-Axis	0.31	0.35	0.33
	Margin	0.29	0.27	0.28
Medium KNN†	Axis	0.52	0.79	0.63
	Off-Axis	0.28	0.17	0.21
	Margin	0.33	0.07	0.11
Coarse KNN†	Axis	0.52	0.99	0.68
	Off-Axis	0.50	0.02	0.04
	Margin	-	0.00	-
Cosine KNN†	Axis	0.58	0.78	0.67
	Off-Axis	0.25	0.20	0.22
	Margin	0.31	0.13	0.19
Cubic KNN†	Axis	0.53	0.79	0.64
	Off-Axis	0.23	0.15	0.18
	Margin	0.57	0.13	0.22
Weighted KNN†	Axis	0.58	0.73	0.64
	Off-Axis	0.28	0.24	0.26
	Margin	0.31	0.17	0.22
AdaBoost Trees†§	Axis	0.60	0.68	0.64
	Off-Axis	0.41	0.39	0.40
	Margin	0.36	0.27	0.31
Bagged Trees†§	Axis	0.59	0.73	0.65
	Off-Axis	0.36	0.33	0.34
	Margin	0.27	0.13	0.18
Subspace Discriminant†§	Axis	0.57	0.85	0.68
	Off-Axis	0.29	0.13	0.18
	Margin	0.47	0.23	0.31
Subspace KNN†§	Axis	0.66	0.69	0.67
	Off-Axis	0.35	0.33	0.34
	Margin	0.34	0.33	0.34
RUSBoosted Trees†§	Axis	0.65	0.49	0.56
	Off-Axis	0.35	0.37	0.36
	Margin	0.17	0.27	0.21

Algorithm	Class	Precision	Recall	F1 Score
Random Forest†§	Axis	0.64	0.84	0.72
	Off-Axis	0.60	0.46	0.52
	Margin	0.60	0.30	0.40
XGBoost †§	Axis	0.69	0.78	0.73
	Off-Axis	0.51	0.46	0.48
	Margin	0.56	0.47	0.51
Neural Network†‡	Axis	0.70	0.79	0.74
	Off-Axis	0.46	0.42	0.44
	Margin	0.42	0.32	0.36

*Unsupervised Learning; †Supervised Learning; §Ensemble Classifier; ‡Deep Learning